TECHNICAL REPORT 70-133 SEPTEMBER 1970
JPL CONTRACT NO. 952695
NGL-21-002-008

F O R M A L

A FORMULA MANIPULATION LANGUAGE

BY

CHARLES K. MESZTENYI

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

TECHNICAL REPORT 70-133 SEPTEMBER 1970
JPL CONTRACT NO. 952695
NGL-21-002-008

F O R M A L

A FORMULA MANIPULATION LANGUAGE

BY

CHARLES K. MESZTENYI

THIS RESEARCH WAS SUPPORTED BY CONTRACT 952695 FROM THE JET PROPULSION LABORATORY OF THE CALIFORNIA INSTITUTE OF TECHNOLOGY AND BY GRANT NGL-21-002-008 FROM THE NATIONAL AERONAUTICS AND SPACE ADMINISTRATION TO THE COMPUTER SCIENCE CENTER OF THE UNIVERSITY OF MARYLAND.

ABSTRACT

THE FORMAL SYSTEM IS AN EXTENSION OF FORTRAN V WHICH PROVIDES THE USER WITH THE CAPABILITY OF PERFORMING SYMBOLIC ALGEBRAIC MANIPULATION. IN THIS SYSTEM, THE VALUE OF A VARIABLE CAN BE A SYMBOLIC ALGEBRAIC EXPRESSION. ARITHMETIC OPERATIONS, DIFFERENTIATION, SPECIAL OPERATIONS AND SIMPLIFICATION OF SYMBOLIC ALGEBRAIC EXPRESSIONS ARE ALL PROVIDED BY THE SYSTEM. THE INTERACTIVE FORMAL SYSTEM IS A SUBSET OF THE FORMAL SYSTEM WHICH IS INDEPENDENT OF FORTRAN V. THIS SYSTEM CAN BE USED FOR SYMBOLIC ALGEBRAIC MANIPULATION IN AN INTERACTIVE ENVIRONMENT SUCH AS TELETYPE.

THE SYSTEM IS IMPLEMENTED FOR THE UNIVAC 1108, AND ITS DETAILED TECHNICAL DOCUMENTATION IS CONTAINED IN TECHNICAL REPORT 70-134.

TABLE OF CONTENTS

1. INTRODUCTION

FORMAL IS A PROGRAMMING SYSTEM WHICH IS AN EXTENSION OF FORTRAN. IT CONSISTS OF A LANGUAGE, A PREPROCESSOR, THE FORTRAN COMPILER ITSELF, AND OBJECT TIME SUBROUTINES. THE PURPOSE OF THE SYSTEM IS TO PROVIDE A CAPABILITY FOR PERFORMING SYMBOLIC ALGEBRAIC MANIPULATIONS IN ADDITION TO THE NUMERICAL CALCULATION CAPABILITY OF FORTRAN. THE SYSTEM IS IMPLEMENTED FOR THE UNIVAC 1108 AS AN EXTENSION OF FORTRAN V. THE DEVELOPMENT OF THE FORMAL SYSTEM WAS INSPIRED BY IBM'S FORMAC SYSTEM, WHICH IS AVAILABLE FOR IBM 7090/94 AND 360.

THE LANGUAGE OF FORMAL, AS AN EXTENSION OF FORTRAN, IS COMPOSED OF STATEMENTS. A STATEMENT CAN BE EITHER A NORMAL FORTRAN STATEMENT OR A FORMAL STATEMENT, ABBREVIATED HEREIN AS 'F-STATEMENT'. OUR AIM IS TO DEFINE THE FORMAT (OR SYNTAX) OF THE F-STATEMENTS DESCRIPTIVELY BY USING SYNTACTICAL RULES SIMILAR TO FORTRAN STATEMENTS WHENEVER THEIR SEMANTICS ARE SIMILAR; E.G., THE FORMAL ASSIGNMENT STATEMENT DIFFERS FROM ITS FORTRAN COUNTERPART ONLY IN THAT IT IS ENCLOSED IN APOSTROPHES. THE INTERFACE BETWEEN FORMAL AND FORTRAN STATEMENTS IS PROVIDED BY THE USE OF FORTRAN NUMERICAL EXPRESSIONS WITH SPECIAL FORMATING IN FORMAL STATEMENTS, AND BY THE OPERATION OF FORTRAN VALUED FUNCTIONS ON FORMAL EXPRESSIONS.

THIS REPORT IS DESIGNED TO GIVE A COMPLETE DESCRIPTION OF THE FORMAL SYSTEM AS IT IS IMPLEMENTED FOR THE UNIVAC 1108. IT IS ASSUMED THAT THE READER IS FAMILIAR WITH FORTRAN, THEREFORE NO ATTEMPT IS MADE TO DESCRIBE ANY FORTRAN FEATURES. CHAPTER 3 PROVIDES THE GENERAL SPECIFICATIONS OF THE EXTENDED LANGUAGE AND IS DESIGNED AS A USER'S MANUAL. CHAPTER 4 CONTAINS EXAMPLES FOR THE EXTENDED LANGUAGE. CHAPTER 5 DESCRIBES THE INTERACTIVE FORMAL SYSTEM, WITH CHAPTER 6 CONTAINING EXAMPLES FOR THE USE OF THIS SYSTEM. CHAPTER 7 LISTS THE ERROR MESSAGES PRINTED BY THE SYSTEM. CHAPTER 8 GIVES THE SYNTACTIC DEFINITION OF THE FORMAL STATEMENTS. CHAPTER 9 DESCRIBES THE INTERNAL STRUCTURE OF THE SYMBOLIC EXPRESSIONS; AND THROUGH THIS DESCRIPTION, GIVES A PRECISE DEFINITION OF THE SPECIAL FORMULA MANIPULATING FUNCTIONS - SUCH AS THE NUMERATOR/DENOMINATOR OF AN EXPRESSION, ETC.

NOTE:
THE INTERACTIVE FORMAL SYSTEM CONSISTS OF ONLY FORMAL STATEMENTS. THUS, IT IS NOT AN EXTENSION OF FORTRAN. THIS SYSTEM IS IMPLEMENTED BY THE USE OF THE OBJECT TIME SUBROUTINES USED BY THE EXTENDED FORMAL SYSTEM.

2. ACKNOWLEDGEMENT

THE AUTHOR OF THIS REPOPT GRATEFULLY ACKNOWLEDGE THE CONTRIBUTIONS OF HIS COLLEAGES :

MR. HANS BREITENLOHNER ANALYZED AND WROTE THE PREPROCESSOR FOR THE FORMAL SYSTEM AND HELPED IN MANY OTHER PARTS OF THE PROJECT WHICH WOULD BE TOO NUMEROUS TO LIST.

MR. ROBERT NUNN AND MR. JOSEPH YEH HELPED THE AUTHOR TO DEVELOP THE OBJECT TIME ROUTINES AND THE DOCUMENTATION. PROOF READING WAS PROVIDED BY MISS GAIL VAN METER.

LAST, BUT NOT LEAST, THE AUTHOR IS VERY GRATEFUL TO MR. JOHN MENARD, ASSOC. DIR., WHOSE ENCOURAGEMENT AND ADMINISTRATICE HELP MADE IT POSSIBLE TO FINISH THE PROJECT IN A RELATIVELY SHORT TIME.

3. GENERAL SPECIFICATIONS

THIS CHAPTER CONTAINS THE INFORMATION NECESSARY FOR THE USER OF THE FORMAL SYSTEM. IT IS SUPPLEMENTED WITH EXAMPLES IN CHAPTER 4, AND WITH A LIST OF THE ERROR MESSAGES IN CHAPTER 7. IN SOME CRITICAL CASES, THE USER MAY WANT TO CONSULT THE MORE PRECISE DEFINITIONS IN CHAPTERS 8 AND 9.

3.1. CONTROL CARD AND CARD FORMAT

A FORMAL PROGRAM MAY CONSIST OF SEVERAL SUBPROGRAMS. EACH SUBPROGRAM MUST START WITH A CONTROL CARD. WHEN A SUBPROGRAM DOES NOT CONTAIN ANY FORMAL STATEMENTS, ITS CONTROL CARD CORRESPONDS TO THE LANGUAGE IN WHICH THE SUBPROGRAM WAS WRITTEN. E.G. A FORTRAN SUBPROGRAM WITHOUT FORMAL STATEMENTS HAS A CONTROL CARD AS FOLLOWS:

```
@FOR,<OPTIONS>    <SPECIFICATIONS>
```

(AS DESCRIBED IN THE EXEC 8 MANUAL). A FORTRAN SUBPROGRAM CONTAINING FORMAL STATEMENTS MUST HAVE THE CONTROL CARD AS BELOW:

```
@FORMAL*LIB.FORMAL,<OPTIONS>    <SPECIFICATIONS>
```

THE OPTIONS USED ON A FORMAL PROCESSOR CALL CARD ARE LISTED BELOW.

B	DO NOT AUTOMATICALLY COMPILE THE GENERATED SYMBOLIC ELEMENT.
I	INPUT A NEW PROGRAM FROM CARDS.
L	PRODUCES A LISTING OF THE SOURCE CODE AND THE GENERATED SOURCE CODE.
N	PRODUCES NO LISTING.
S	PRODUCES A LISTING OF THE ORIGINAL SOURCE CODE ONLY.
U	UPDATE THE SOURCE INPUT ELEMENT.
W	LIST SOURCE MODIFICATION CARDS WITH SOURCE PROGRAM LISTING.

THE SPECIFICATION FIELDS MUST BE WRITTEN IN ELEMENT NOTATION (SEE PRM 5.6.1).

IF THE 'I' OPTION IS PRESENT, SPEC 1 SPECIFIES THE NAME OF THE SYMBOLIC ELEMENT TO BE GENERATED FROM THE INPUT CARDS (FOR POSSIBLE LATER UPDATING). IF SPEC 1 IS NOT GIVEN, THE SOURCE IMAGES WILL NOT BE SAVED. IF SPEC 2 IS NOT GIVEN, SPEC 1 WILL BE USED FOR IT.

IF THE 'I' OPTION IS ABSENT, SPEC 1 SPECIFIES THE SYMBOLIC ELEMENT TO BE USED FOR SOURCE INPUT, SPEC 3 SPECIFIES AN UPDATED SYMBOLIC ELEMENT TO BE GENERATED BY APPLYING THE CORRECTIONS (SEE PRM 5.6.3) TO SPEC 1. IN THIS CASE SPEC 2 MUST BE SPECIFIED.

SPEC 2 ALWAYS DENOTES THE NAME OF A SYMBOLIC ELEMENT WITH THE TRANSLATED IMAGES, WHICH WILL BE GENERATED BY THE PREPROCESSOR.
IF THE 'B' OPTION IS NOT SPECIFIED, THE PREPROCESSOR WILL DYNAMICALLY ADD THE STATEMENTS :

```
@RALPH,FS    <SPEC 2>,<SPEC 4>
@EOF
```

IF THE <SPEC 4> IS NOT GIVEN, THE RALPH COMPILER WILL AUTOMATICALLY USE <SPEC 2> FOR THE RELOCATABLE OUTPUT FIELD.
IF THE 'B' OPTION IS SET, THE USER HAS TO COMPILE THE GENERATED SYMBOLIC ELEMENT.

BEFORE THE EXECUTION, THE PROGRAMS WILL HAVE TO BE COLLECTED USING THE FOLLOWING SEQUENCE :

```
@MAP
LIB     FORMAL*LIB.
@XQT
.
DATA CARDS
.
```

THE CARD FORMATS OF FORMAL STATEMENTS ARE ESSENTIALLY THE SAME AS THOSE OF THE FORTRAN STATEMENTS. COLUMNS 1-5 ARE RESERVED FOR STATEMENT NUMBER, COLUMN 6 IS FOR CONTINUATION, THE STATEMENT MUST APPEAR BETWEEN COLUMNS 7 AND 72, AND COLUMNS 73-80 MAY CONTAIN IDENTIFICATION (SEQUENCE NUMBERING). BLANK COLUMNS CAN BE FREELY INTERSPERSED IN THE STATEMENT.

3.2. FORMAL STATEMENTS

A FORMAL STATEMENT, OR F-STATEMENT, IS EITHER LABELED OR UNLABELED. THE LABEL IS A STATEMENT NUMBER SIMILAR TO THAT OF A FORTRAN STATEMENT. SINCE THERE ARE NO NON-EXECUTABLE FORMAL STATEMENTS, THE FORTRAN STATEMENTS, 'DO', 'GO TO', MAY REFER TO THE STATEMENT NUMBER OF AN F-STATEMENT IN THE PROPER FORTRAN CONTEXT.

F-STATEMENTS ARE DISTINGUISHED FROM FORTRAN STATEMENTS IN THAT THEIR FIRST AND LAST CHARACTERS ARE APOSTROPHES, AND NO APOSTROPHE MAY APPEAR INSIDE THE STATEMENTS. THERE ARE 6 TYPES OF F-STATEMENTS: F-ASSIGN, F-READ, F-WRITE, F-OPTION, F-ERASE AND

F-DUMP. THE F-ASSIGN STATEMENT ASSIGNS A VALUE, A SYMBOLIC ALGEBRAIC EXPRESSION, TO A FORMAL VARIABLE. THE F-READ AND F-WRITE STATEMENTS PROVIDE SYMBOLIC EXPRESSIONS TO BE READ FROM CARDS AND WRITTEN ON A FORTRAN BCD OUTPUT UNIT. THE F-OPTION STATEMENT DEFINES DIFFERENT OPTIONS DURING EXECUTION SUCH AS THE USE OF THE DISTRIBUTIVE LAW IN EXPRESSION MANIPULATIONS. THE F-ERASE STATEMENT RESETS THE VALUES OF VARIABLES TO THEIR NAMES. THE F-DUMP STATEMENT OUTPUTS SYMBOLIC EXPRESSIONS AND SYMBOL TABLES IN THEIR INTERNAL FORMAT.

3.2.1. F-CONSTANTS

CONSTANTS IN FORMAL STATEMENTS MAY APPEAR EITHER IN DECIMAL FORM OR IN THE FORM OF FORTRAN EXPRESSIONS. IN EITHER FORM, A CONSTANT MAY BE EITHER AN INTEGER OR A SINGLE PRECISION REAL (FLOATING POINT) NUMBER.

THE FORM OF DECIMAL CONSTANTS IS THE SAME AS THAT IN THE FORTRAN STATEMENTS, THUS:

0, 22, -2001

DEFINE DECIMAL INTEGERS, WHILE

.002, -10.2, 23.8E-2

DEFINE REAL CONSTANTS.

DURING EXECUTION, FORTRAN EXPRESSIONS YIELD NUMERICAL VALUES. THESE NUMERICAL VALUES MAY BE USED IN FORMAL STATEMENTS AS CONSTANTS. A MAXIMUM OF 20 FORTRAN EXPRESSIONS MAY APPEAR IN ONE FORMAL STATEMENT. HOWEVER, ONLY INTEGER AND REAL VALUED FORTRAN EXPRESSIONS ARE PERMITTED TO BE USED, AND OBVIOUSLY NO MIXED MODE OPERATION IS ALLOWED. THE PROPER USE OF FORTRAN EXPRESSIONS IN FORMAL STATEMENTS IS TO ENCLOSE THEM IN A PAIR OF PARENTHESES AND PRECEDE THEM WITH A SPECIAL CHARACTER: #, FOLLOWED BY I OR R SUCH THAT #I(...) INDICATES AN INTEGER, #R(...) INDICATES A REAL CONSTANT. E.G.

```
DO 10 I=1,N
X=...
Y=...
'...#R(X)...#R(X*Y)...#I(N*I-1)...'
```

IN THE ABOVE FORMAL STATEMENT, THE 3 FORTRAN VALUED EXPRESSIONS ARE X, X*Y, AND N*I-1, WITH THE FIRST TWO BEING REAL, THE LAST INTEGER.

DURING EXECUTION, THE FORMAL SYSTEM IS CAPABLE OF HANDLING RATIONAL CONSTANTS AS RATIOS OF INTEGERS. SINCE RATIONAL NUMBERS

IN FORMAL STATEMENTS ALWAYS APPEAR CONCURRENTLY WITH OPERATORS, E.G. 3/5 OR 3/(5*X)=(3/5)/X, THEY ARE NOT PART OF THE LANGUAGE ITSELF, BUT THEY ARE PART OF THE IMPLEMENTATION OF THE LANGUAGE.

3.2.2. F-VARIABLES

VARIABLES APPEARING IN FORMAL STATEMENTS ARE REFERED TO AS F-VARIABLES. THE RULES FOR NAMING F-VARIABLES ARE THE SAME AS THOSE FOR NAMING FORTRAN VARIABLES EXCEPT THAT THERE IS NO IMPLICIT ARITHMETIC MODE (INTEGER OR REAL) ASSOCIATED WITH THE FIRST LETTER OF THE VARIABLE NAME.

F-VARIABLES MAY BE SUBSCRIPTED WITH A MAXIMUM OF 4 SUBSCRIPTS. THERE IS NO DECLARATION ASSOCIATED WITH THE SUBSCRIPTED F-VARIABLES, BUT THE SUBSCRIPTS MUST BE INTEGER CONSTANTS, INTEGER VALUED FORTRAN EXPRESSIONS, OR F-EXPRESSIONS WHOSE VALUES ARE INTEGERS. FURTHERMORE, THE VALUE OF A SUBSCRIPT MUST BE BETWEEN 0 AND 511, INCLUSIVELY.

DURING EXECUTION, THE F-VARIABLES ARE CLASSIFIED AS EITHER ATOMIC OR ASSIGNED. THE VALUE OF AN ATOMIC F-VARIABLE IS ITS NAME. ANY F-VARIABLE IS ASSUMED TO BE ATOMIC UNTIL AN F-ASSIGN STATEMENT IS EXECUTED IN WHICH THE VARIABLE APPEARS ON THE LEFT SIDE OF THE STATEMENT.

EXAMPLES:

```
X, A12B, RS(2), Y(0,2,#I(3*I),511)
```

3.2.3. F-EXPRESSIONS

F-EXPRESSIONS ARE CONSTRUCTED IN THE SAME MANNER AS FORTRAN ARITHMETIC EXPRESSIONS EXCEPT THAT THE FUNCTION ARGUMENTS ARE ENCLOSED IN BRACKETS INSTEAD OF PARENTHESES. THUS A FORMAL EXPRESSION CONSISTS OF CERTAIN SEQUENCES OF CONSTANTS, SUBSCRIPTED AND NON-SUBSCRIPTED VARIABLES, AND FUNCTION REFERENCES SEPARATED BY ARITHMETIC OPERATORS, COMMAS, AND PARENTHESES.

THE ARITHMETIC OPERATORS ARE THE SAME AS THE FORTRAN ARITHMETIC OPERATORS, I.E.

+	ADD OPERATOR	(UNARY OR BINARY)
-	SUBTRACT OPERATOR	(UNARY OR BINARY)
*	MULTIPLY OPERATOR	(BINARY)
/	DIVIDE OPERATOR	(BINARY)
**	EXPONENTIAL OPERATOR	(BINARY)

SOME OF THESE OPERATORS ARE TRANSFORMED BY THE FORMAL PROCESSOR; THERE ARE NO INTERNAL REPRESENTATION OF THE SUBTRACT (-) AND

DIVIDE (/) OPERATORS, AND THE UNARY ADD OPERATOR IS SIMPLY DISREGARDED. IF E REPRESENTS ANY F-EXPRESSION, THEN THESE TRANSFORMATIONS ARE AS FOLLOWS:

```
+E      =>   E

-E      =>   (-1)*E

E1-E2   =>   E1+(-1)*E2

E1/E2   =>   E1*E2**(-1)
```

THE FUNCTIONAL REFERENCES HAVE THE FORM OF A FUNCTION IDENTIFIER FOLLOWED BY ITS ARGUMENTS WHICH ARE SEPARATED BY COMMAS AND ENCLOSED IN BRACKETS. FOR EXAMPLE,

```
DIF[X*A,X,2]
```

IS A FUNCTION REFERENCE TO THE DIF (DIFFERENTIAL OPERATION) FUNCTION WITH 3 ARGUMENTS. THE ARGUMENTS OF CERTAIN FUNCTIONS MAY CONTAIN FURTHER FUNCTION REFERENCES, E.G.

```
NUM[DIF[X*A,X,2]]
```

THERE ARE 5 TYPES OF FUNCTIONS AVAILABLE IN THE FORMAL STATEMENTS: MATHEMATICAL, SPECIAL OR FORMAL, DIFFERENTIAL, DEFINED, AND UNDEFINED FUNCTIONS. THEY ARE DESCRIBED IN THE SUBSEQUENT SECTIONS OF THIS MANUAL.

IN THE HIERARCHY OF OPERATIONS, PARENTHESES MAY BE USED IN THE F-EXPRESSIONS TO SPECIFY THE ORDER IN WHICH OPERATIONS ARE TO BE EVALUATED, AS IN FORTRAN ARITHMETIC EXPRESSIONS. E.G.

```
2*X*(Y+#I(3*I)*Z)+X**N

X+SUBST[A*(B+C),Z,3.5,SIN[Y],0.5]
```

3.2.4. MATHEMATICAL FUNCTIONS

THE FOLLOWING MATHEMATICAL FUNCTIONS ARE AVAILABLE:

EXP[X]	EXPONENTIAL FUNCTION
LOG[X]	LOGARITHM FUNCTION
TNH[X]	HYPERBOLIC TANGENT
SIN[X]	SIN FUNCTION
COS[X]	COS FUNCTION
ATN[X]	ARCTAN FUNCTION
FAC[X]	FACTORIAL FUNCTION, F[N] = N!
BIN[X,Y]	BINOMIAL COEFFICIENT, $BIN[N,K] = \binom{N}{K}$
STEP[X,Y,Z]	STEP FUNCTION

$$STEP[X,Y,Z] = \begin{cases} 1 & \text{IF } X=<Y<Z \text{ OR } X=Y=Z \\ 0 & \text{OTHERWISE} \end{cases}$$

WHERE X,Y,Z MAY BE ANY F-EXPRESSIONS. THE LAST 3 FUNCTIONS, FAC, BIN, AND STEP, ARE REFERED TO AS INTEGER VALUED FUNCTIONS.

WHEN THE ARGUMENTS ARE CONSTANTS, THE VALUES OF THE FUNCTIONS ARE EVALUATED WITH REGARD TO THE OPTIONS MFCT/NOMFCT, INT/NOINT:

OPTION MFCT --- EXP,LOG,TNH,SIN,COS,TAN FUNCTIONS ARE EVALUATED WITH CONSTANT ARGUMENT.

NOMFCT --- THE ABOVE FUNCTIONS ARE LEFT AS FUNCTIONAL EXPRESSIONS.

INT --- FAC[N] IS EVALUATED WHEN N IS AN INTEGER CONSTANT, BIN[N,K] IS EVALUATED FOR INTEGER CONSTANTS N AND K, STEP[X,Y,Z] IS EVALUATED FOR CONSTANTS X,Y,Z.

NOINT --- FAC,BIN AND STEP FUNCTIONS ARE LEFT AS FUNCTIONAL EXPRESSIONS.

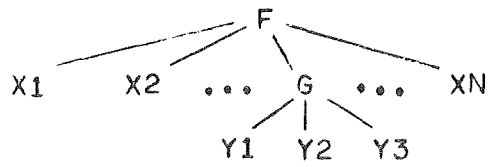
FOR SETTING THE OPTIONS, SEE SECTION 3.2.11.

3.2.5. FORMAL SPECIAL FUNCTIONS

TO USE THE FORMAL SPECIAL FUNCTIONS, ONE MUST FIRST UNDER-

STAND THE INTERNAL REPRESENTATION OF EXPRESSIONS. THE EXPRESSIONS ARE STORED IN PREFIX NOTATION DURING THE EXECUTION. THIS NOTATION IS A LEFT-TO-RIGHT LINEARIZATION OF THE TREE REPRESENTATION OF THE EXPRESSIONS.

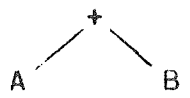
THE TREE REPRESENTATION OF AN ARBITRARY EXPRESSION CONSISTING OF OPERATORS, F, APPLIED TO OPERANDS, X1,X2,...,XN, IS A TREE WITH F AS THE ROOT AND X1, X2, ..., XN AS TERMINAL VERTICES. HOWEVER, IF ANY ARGUMENT, XI, IS ITSELF AN EXPRESSION, G(Y1,Y2,Y3), THEN XI IS REPRESENTED BY A SUBTREE:



ALL TERMINAL VERTICES ARE EITHER CONSTANTS OR ATOMIC VARIABLES. ALL NON-TERMINAL VERTICES ARE OPERATORS. THE NUMBER OF ARGUMENTS OF AN OPERATOR IS THE NUMBER OF OUTGOING VERTICES. THE ROOT OF A GIVEN TREE IS REFERRED TO AS THE LEAD OPERATOR OF THE CORRESPONDING EXPRESSION.

THE ARITHMETIC OPERATORS CAN BE REPRESENTED IN FUNCTION NOTATION:

A+B => +(A,B) =>



A*B => *(A,B) =>

A**B => **(A,B) =>

WHEN MORE THAN 2 TERMS ARE ADDED OR MULTIPLIED, THE FORMAL PROCESSOR AUTOMATICALLY COMBINES THE ARGUMENTS UNDER ONE VERTEX:

A+B+C => +(A,B,C) =>
A*B*C => *(A,B,C)

SINCE + AND * ARE SYMMETRIC OPERATORS

+(A,B) = +(B,A) , *(A,B) = *(B,A)

THE FORMAL PROCESSOR AUTOMATICALLY ORDERS THE ARGUMENTS LEXICOGRAPHICALLY, THAT IS ACCORDING TO THE FOLLOWING SCHEME:

CONSTANTS

VARIABLES IN ALPHABETIC ORDER

SUBEXPRESSIONS ACCORDING TO THEIR LEAD OPERATORS:

+ (ADDITION)

* (MULTIPLICATION)

** (EXPONENTIATION)

MATHEMATICAL FUNCTIONS

DIFFERENTIAL OPERATOR

UNDEFINED FUNCTIONS IN ALPHABETIC ORDER

IF TWO SUBEXPRESSIONS HAVE THE SAME LEAD OPERATOR, THEN THE RELATION IS ESTABLISHED BY COMPARING THEIR NUMBERS OF ARGUMENTS. IF THESE STILL AGREE, THEN THE ARGUMENTS THEMSELVES ARE COMPARED.

THE AVAILABLE SPECIAL FUNCTIONS ARE AS FOLLOWS:

EXPAND[E]

WHERE E IS AN F-EXPRESSION. THIS FUNCTION TRANSFORMS E TO AN EQUIVALENT FORM BY THE APPLICATION OF THE DISTRIBUTIVE LAW OF MULTIPLICATION:

X*(Y+Z) => X*Y+X*Z AND (X+Y)*Z => X*Z+Y*Z

FURTHERMORE, SUMS RAISED TO INTEGER POWERS ARE EXPANDED BY THE APPLICATION OF THE BINOMIAL LAW:

(X+Y)**K => X**K + BIN[K,1]*X**(K-1)*Y + ... + Y**K

THE EXPRESSION E IS CHECKED FOR THE APPLICATION OF THESE TRANSFORMATIONS IN ALL LEVELS.

EXAMPLE: E = (X+Y)*(X-Y) + SIN[2*(X+Y)]

EXPAND[E] = X**2-Y**2+SIN[2*X+2*Y]

FLOAT[E]
FLOATB[E]

WHERE E IS AN F-EXPRESSION. THESE FUNCTIONS CONVERT THE INTEGER AND RATIONAL CONSTANTS IN E INTO THEIR EQUIVALENT REAL FORM. FLOAT CONVERTS ALL CONSTANTS IN E. FLOATB CONVERTS ONLY THOSE CONSTANTS WHICH ARE NOT IN EXPONENTS.

EXAMPLE:
FLOAT[2*X**1/2] = 2.0 * X**0.5
FLOATB[2*X**1/2] = 2.0 * X**1/2

FIX[E,N]
FIXE[E,N]

WHERE E IS AN F-EXPRESSION AND N IS AN INTEGER CONSTANT OR AN F-EXPRESSION WHOSE VALUE IS AN INTEGER. THESE FUNCTIONS CONVERT REAL CONSTANTS IN E INTO INTEGERS. THE CONVERSION IS DONE AS FOLLOWS:

LET R BE A REAL CONSTANT, THEN DEFINE I AS THE NEAREST INTEGER TO R. THEN R IS REPLACED BY I IF AND ONLY IF THE ABSOLUTE VALUE OF THE DIFFERENCE BETWEEN R AND I IS LESS THAN OR EQUAL TO 10**N. NOTE THAT N SHOULD BE A NEGATIVE INTEGER.

FIX CONVERTS ALL REAL CONSTANTS IN E SUBJECT TO THE ABOVE CONDITION. FIXE CONVERTS REAL CONSTANTS ONLY IF THEY ARE PART OF AN EXPONENT.

EXAMPLE:
FIX[2.1*X**2.2,-1] = 2*X**2.2
FIX[2.1*X**2.2,0] = 2*X**2

PRODEX[E]

THIS FUNCTION DISTRIBUTES THE EXPONENT OF A PRODUCT AS BASE OVER ITS FACTORS, I.E

(E1*E2*...*EN)**F => (E1**F)*(E2**F)*...*(EN**F)

ARG[E,N]

WHERE E IS AN F-EXPRESSION AND N IS A POSITIVE INTEGER CONSTANT. THIS FUNCTION EXTRACTS THE N'TH ARGUMENT OF THE LEAD OPERATOR OF E.

EXAMPLE: E = 2+3*X+4*X**2

```
ARG[E,3]=4*X**2,
ARG[ARG[E,3],2]=X**2,
ARG[ARG[ARG[E,3],2],2]=2,
```

```
  NUM[E]
  ------
  DENOM[E]
```

WHERE E IS AN F-EXPRESSION. THESE FUNCTIONS EXTRACT THE NUMERATOR AND DENOMINATOR PART OF E, RESPECTIVELY. OBVIOUSLY, NUM[E]/DENOM[E]=E. THESE FUNCTIONS MUST BE CLARIFIED BY THE FOLLOWING DEFINITION FOR THE NUMERATOR/DENOMINATOR OF AN F-EXPRESSION E:

1. E IS NOT A PRODUCT OR EXPONENTIAL EXPRESSION, I.E. THE LEAD OPERATOR OF E IS NOT * OR **, THEN
   NUM[E] = E DENOM[E] = 1

2. E IS AN EXPONENTIAL EXPRESSION, I.E. THE LEAD OPERATOR OF E IS **, SO THAT E=A**B, THEN IF B IS NOT A SUM, THEN
   EITHER NUM[E]=1, DENOM[E]=E, IF B IS A NEGATIVE CONSTANT OR B IS A PRODUCT OF A NEGATIVE CONSTANT AND AN ARBITRARY F-EXPRESSION
   OR NUM[E]=E, DENOM[E]=1, IF B DOES NOT SATISFY THE ABOVE CONDITION.
   IF B IS A SUM, B=B1+...+BM, THEN E=(A**B1)*...*(A**BM), THE NUMERATOR AND DENOMINATOR OF E IS DETERMINED AS E HAS BEEN A PRODUCT.

3. E IS A PRODUCT, I.E. THE LEAD OPERATOR OF E IS *, THEN
   E=E1*...*EN
   AND NUM[E]=NUM[E1]*...*NUM[EN]
   DENOM[E]=DENOM[E1]*...*DENOM[EN]

EXAMPLES:
```
NUM[5+X**(-1)] = 5+X**(-1)

NUM[X**(N-5)] = X**(N-5)

NUM[X**(-5*N)] = 1

NUM[3*(X**(-2))*Y]  = 3*Y
```

BASE[E]

EXPON[E]

WHERE E IS AN F-EXPRESSION. THESE FUNCTIONS DELIVER THE BASE AND EXPONENTIAL PARTS OF E PROVIDED THAT THE LEAD OPERATOR OF E IS **. OTHERWISE BASE[E]=E AND EXPON[E]=1.

EXAMPLES: BASE[(X+Y)**(5-N)]=X+Y

BASE[5*X**2]=5*X**2, EXPON[5*X**2]=1

BASE[-X**2]=-X**2 =(-1)*X**2

BASE[ARG[-X**2,2]] = BASE[X**2] = X

CODEM[E]

WHERE E IS AN F-EXPRESSION. THIS FUNCTION TRANSFERS E INTO A COMMON DENOMINATOR FORM. IF E IS NOT A SUM, I.E. THE LEAD OPERATOR OF E IS NOT +, THEN NO TRANSFORMATION TAKES PLACE. OTHERWISE, IF E=E1+...+EN, THE DENOMINATORS OF EACH TERM, DENOM[E1],..., DENOM[EN], ARE DETERMINED ACCORDING TO THE DEFINITION OF THE DENOM FUNCTION. THESE DENOMINATORS ARE EITHER 1 OR EXPONENTIAL EXPRESSIONS OF THE FORM AI**BI. THE LEAST COMMON MULTIPLE, CD, OF THESE EXPONENTIAL EXPRESSIONS IS DETERMINED AS THE PRODUCT OF THOSE DENOMINATORS, AI**BI, FOR WHICH THERE EXISTS NO OTHER DENOMINATOR AJ**BJ, I≠J, SUCH THAT AI=AJ AND BI<BJ. THE RELATION BI<BJ IS THE NATURAL RELATION IF BOTH OF THEM ARE CONSTANTS, OR BI=C1*D AND BJ=C2*D WITH THE CONSTANT C1 AND C2 SATISFYING THE RELATION C1<C2. ONCE THE LEAST COMMON MULTIPLE, OR COMMON DENOMINATOR, CD IS DETERMINED, E=E1+E2+...+EN IS TRANSFORMED TO (E1*CD+E2*CD+...+EN*CD)*(CD**(-1)).

EXAMPLES: CODEM[X*Y**(-1)] = X*Y**(-1)

CODEM[X+Y**(-1)] = (X*Y+1)*(Y**(-1))

CODEM[X**(-2*N)+X**(-3*N)]= (X**N+1)*(X**(-3N))

COEFF[E,X]

WHERE E IS AN F-EXPRESSION, AND X IS AN F-EXPRESSION SUCH THAT IT IS NEITHER A SUM NOR A PRODUCT, I.E. THE LEAD OPERATOR OF X IS NEITHER + NOR *. THIS FUNCTION EXTRACTS THE COEFFICIENT OF X IN E WHICH IS DETERMINED AS FOLLOWS:

1. IF E IS NEITHER A SUM NOR A PRODUCT, I.E. THE LEAD OPERATOR OF E IS NEITHER + OR *, THEN COEFF[E,X]=1 IF E=X, AND COEFF[E,X]=0 IF E≠X.

2. IF E IS A PRODUCT, I.E. E=E1*E2*...*EN, THEN COEFF[E,X]=E/EJ IF THERE EXISTS J SUCH THAT EJ=X FOR SOME 1=<J=<N, COEFF[E,X]=0 OTHERWISE.

3. IF E IS A SUM, I.E. E=E1+E2+...+EN, THEN COEFF[E,X]=COEFF[E1,X]+ COEFF[E2,X]+...+COEFF[EN,X].

EXAMPLES: COEFF[E,E] = 1

COEFF[2+3*X+4*X**2,X**2] = 4

COEFF[X+X*SIN[X],X] = 1+SIN[X]

SUBST[E,X1,Y1,...,XN,YN] (N<21)

WHERE E,X1,Y1,...,XN,YN ARE ARBITRARY F-EXPRESSIONS. IN THE EXPRESSION E, Y1,Y2,...,YN ARE SUBSTITUTED FOR X1,X2,...,XN, RESPECTIVELY. THE SUBSTITUTION OCCURS IN A LOOP; FIRST Y1 IS SUBSTITUTED FOR X1 IN E, THEN Y2 IS SUBSTITUTED FOR X2 IN THE PREVIOUSLY CHANGED EXPRESSION E, ETC.

EXAMPLE: SUBST[A*Z+Z**2,Z,X+Y,Y,3]
= SUBST[A*(X+Y)+(X+Y)**2,Y,3]
= A*(X+3)+(X+3)**2

REPLAC[E,X1,Y1,...,XN,YN] (N<21)

THIS FUNCTION IS SIMILAR TO SUBST, BUT THE SUBSTITUTION DOES NOT OCCUR IN A LOOP. THUS, THIS FUNCTION MAY BE USED TO EXCHANGE VARIABLES.

EXAMPLE: REPLAC[X-Y,X,Y,Y,X] = Y-X

REPLAC[X**2+X**4,X**2,-1,X**4,1] = 0

WHEN THE EXPRESSION XI TO BE REPLACED BY YI IN SUBST OR REPLAC FUNCTIONS, IS A SUM OR PRODUCT, THEN THE FOLLOWING RULES APPLY: IF XI IS A SUM, THEN IT IS REPLACED ONLY IF IT APPEARS WITHOUT DISTRIBUTED FACTORS:

IF XI=A+D THEN
C*(A+B+D) => C*(B+YI)
BUT C*A+C*B+C*D REMAINS UNCHANGED.

IF XI IS A PRODUCT, THEN IT IS REPLACED ONLY IF IT APPEARS WI-

THOUT DISTRIBUTED EXPONENTS:
IF XI=A*D THEN
(A*B*D)**C => (B*YI)**C
BUT (A**C)*(D**C) REMAINS UNCHANGED.

REPLACEMENT OF A SUBEXPRESSION IS PERFORMED BY THE REPLAC FUNTION ACCORDING TO THE ORDER OF ITS ARGUMENTS. THUS
REPLAC[A+B+C,A+C,X,A+B,Y] = B+X
REPLAC[A+B+C,A+B,Y,A+C,X] = C+Y

NOTE: SUBST[E,X1,Y1,X2,Y2] = REPLAC[REPLAC[E,X1,Y1],X2,Y2]

LDOP[E]

NARG[E]

WHERE E IS AN F-EXPRESSION. THESE FUNCTIONS ARE INTEGER VALUED FUNCTIONS. LDOP[E] GIVES THE IDENTIFIER INTEGER OF THE LEAD OPERATOR OF E. NARG[E] GIVES THE NUMBER OF ARGUMENTS OF THE LEAD OPERATOR OF E. THESE SPECIAL FUNCTIONS CORRESPOND TO THE SIMILAR FORTRAN VALUED FUNCTIONS, SEE SECTION 3.3.

REMARK:
THE FOLLOWING 6 SPECIAL FUNCTIONS:
ARG
NUM
DENOM
BASE
EXPON
COEFF
ARE NOT EVALUATED WHENEVER THEIR ARGUMENTS CONTAIN A DUMMY VARIABLE OF A DEFINED FUNCTION, SEE 3.2.7.

3.2.6. DIFFERENTIAL FUNCTION

THE DIFFERENTIAL FUNCTION DIFFERENTIATES AN EXPRESSION IN RESPECT TO A VARIABLE. ITS FORM IS AS FOLLOWS

$$DIF[E,X,N] = \frac{\partial^N E}{\partial X^N}$$

WHERE E IS ANY F-EXPRESSION, X MUST BE A VARIABLE (SUBSCRIPTED OR NOT) AND N MUST BE A POSITIVE INTEGER CONSTANT WHICH GIVES THE ORDER OF DIFFERENTIATION.

EXAMPLE: `DIF[X*SIN[X*Y],X,2] = 2*Y*COS[X*Y]-X*Y**2*SIN[X*Y]`

WHEN THE EXPRESSION, E, CONTAINS AN UNDEFINED FUNCTION WITH ITS ARGUMENTS CONTAINING THE DIFFERENTIAL VARIABLE OR AN UNDEFINED ARGUMENT OF A DEFINED FUNCTION, THEIR DIFFERENTIATION WILL BE ONLY NOTED AND NOT EXECUTED. THE DIFFERENTIATION IS DENOTED BY '(Δ/ΔX)' IN OUTPUT LISTINGS.

WHEN THE ARGUMENTS OF AN UNDEFINED FUNCTION DO NOT CONTAIN THE DIFFERENTIAL VARIABLE, THEN THEIR DIFFERENTIAL QUOTIENTS ARE ZERO. FOR THE DEFINITION OF UNDEFINED FUNCTION, SEE 3.2.8.

3.2.7. DEFINED FUNCTIONS

FREQUENTLY USED FUNCTIONS MAY BE DEFINED BY AN ASSIGN STATEMENT SIMILAR TO FORTRAN FUNCTION STATEMENTS. BUT UNLIKE THOSE STATEMENTS, THE ARGUMENTS (DUMMY VARIABLES) MUST BE DENOTED BY POSITIVE INTEGERS IN BRACKETS SUCH THAT AN INTEGER N REFERS TO THE N'TH ARGUMENT. E.G.

```
F = [3]*(A*[1]**2+B*[2]**2)
```

DEFINES A FUNCTION F OF 3 VARIABLES.

THE NAME OF THE DEFINED FUNCTION MAY BE SUBSCRIPTED FOLLOWING THE RULES OF SUBSCRIPTION FOR VARIABLES.

THE FOLLOWING 6 SPECIAL FUNCTIONS: ARG, NUM, DENOM, BASE, EXPON AND COEFF, ARE NOT EVALUATED WHEN THEIR ARGUMENTS CONTAIN DUMMY VARIABLES, E.G.

```
FN = NUM[[1]/[2]]
```

IS LEFT UNCHANGED.

ONCE A FUNCTION IS DEFINED BY AN ASSIGN STATEMENT IN THE ABOVE FASHION OR BY A READ STATEMENT, IT MAY BE USED WITH THE PROPER NUMBER OF ARGUMENTS. THE ARGUMENTS MAY BE ANY F-EXPRESSIONS. FOR EXAMPLE, THE ABOVE FUNCTION MAY BE REFERENCED AS

```
F[X+Y,X-Y,Z]
```

WHICH GIVES THE FOLLOWING EXPRESSION

```
Z*(A*(X+Y)**2+B*(X-Y)**2)
```

WHEN THE DEFINED FUNCTION HAS UNEVALUATED SPECIAL FUNCTIONS, THEY WILL BE EVALUATED AFTER SUBSTITUTION OF THE DUMMY VARIABLES. THUS

```
FNC X/Y, Y**2/Z]
```

GIVES

```
X*Z
```

A FUNCTION MAY NOT BE DEFINED RECURSIVELY. E.G.

```
G = [1]*G[X-1]
```

A FUNCTION MAY BE REDEFINED BY OTHER ASSIGN OR READ STATEMENTS DURING EXECUTION.

3.2.8. UNDEFINED FUNCTIONS

IF A FUNCTION IS REFERENCED BY AN IDENTIFIER FOLLOWED BY ARGUMENTS IN BRACKETS, AND IF IT WAS NOT DEFINED PREVIOUSLY BY AN ASSIGN OR READ STATEMENT, THEN THE FUNCTION IS AN UNDEFINED FUNCTION. THE ARGUMENTS CAN BE ARBITRARY F-EXPRESSIONS.

THE NUMBER OF ARGUMENTS OF AN UNDEFINED FUNCTION MAY NOT EXCEED SEVEN.

3.2.9. ASSIGN STATEMENT

A FORMAL ASSIGN STATEMENT MAY DEFINE A FUNTION OR THE VALUE OF A VARIABLE.

WHEN AN ASSIGN STATEMENT DEFINES A FUNCTION F,

```
F = E = ...[1]...
```

THE DEFINING EXPRESSION E, FOLLOWING THE = CHARACTER, MUST CONTAIN AT LEAST ONE ARGUMENT REFERENCE, I.E. POSITIVE INTEGER IN BRACKETS AS AN OPERAND. THE LARGEST APPEARING INTEGER DEFINES THE NUMBER OF ARGUMENTS OF THE FUNCTION, AND AN INTEGER N REFERS TO THE N'TH ARGUMENT. THE FUNCTION ASSIGN STATEMENT IS NOT RECURSIVE, I.E. THE DEFINING EXPRESSION E OF THE FUNCTION F MAY NOT REFERENCE ITSELF, DIRECTLY OR INDIRECTLY. FOR FURTHER USE OF THE ONCE DEFINED FUNCTION, SEE SECTION 3.7.

THE FORMAL ASSIGN STATEMENT DEFINES OR REDEFINES THE VALUE OF A VARIABLE SIMILARLY TO A FORTRAN ASSIGN STATEMENT:

V = E OR W = E1,E2,...,EN

WHERE V,W ARE F-VARIABLES, POSSIBLY SUBSCRIPTED, AND E,E1,...,EN ARE F-EXPRESSIONS. IN THE LATER CASE, W IS CALLED A LIST-VARIABLE AND MAY BE USED TO REPRESENT A LIST OF FUNCTION ARGUMENTS. E.G.:

```
W(1) = X,1,Y,2      AND      W(2) = X,2,Y,1
V = SUBST[F[X,Y],W(#I(I))]
```

DEFINES THE VALUE OF THE VARIABLE V AS F[1,2] IF THE FORTRAN VARIABLE I IS ONE, OR AS F[2,1] WHEN I IS TWO.

3.2.10. I/O STATEMENTS

THE FORMAL READ STATEMENT

READ (K) V1,V2,...,VN (N=>1)

READS N F-EXPRESSIONS FROM FORTRAN CARD INPUT UNIT K AS THE VALUES OF VARIABLES OR AS THE DEFINING EXPRESSIONS OF FUNCTIONS, V1,V2,...,VN. EACH F-EXPRESSION ON THE CARDS MUST BE TERMINATED BY SEMICOLON (;). FURTHERMORE, THE CARDS CONTAINING THE F-EXPRESSIONS MAY NOT HAVE ANY OTHER PUNCTUATIONS. THE USE OF BLANK COLUMNS AND THE CONTINUATION OF AN F-EXPRESSION ON THE FOLLOWING CARD IS PERMITTED. THE MAXIMUM NUMBER OF CARDS CONTAINING ONE F-EXPRESSION IS 8.

THE FORMAL WRITE STATEMENT

WRITE (K,L,M) V1,V2,...,VN (N=>1)

WRITES OUT THE VALUES OF THE VARIABLES (OR DEFINING EXPRESSIONS OF FUNCTIONS) V1,...,VN ON THE FORTRAN PRINT OR PUNCH UNIT K. EACH VALUE/EXPRESSION, EI, PRINTED OR PUNCHED IS PRECEDED BY THE NAME OF THE VARIABLE/FUNCTION AND BY AN EQUAL SIGN:

VI = EI

FURTHERMORE, EACH ITEM, VI=EI, STARTS ON A NEW LINE IN COLUMN L AND THE NUMBER OF COLUMNS USED DOES NOT EXCEED M. THUS, THE SUM OF L AND M MAY NOT EXCEED THE AVAILABLE NUMBER OF COLUMNS OF ONE LINE/CARD.

3.2.11. OPTION STATEMENT

THE OPTION STATEMENT DEFINES THE USE OR NON-USE OF CERTAIN EQUIVALENT TRANSFORMATIONS ON SYMBOLIC EXPRESSIONS DURING EXECUTION. IT IS AN EXECUTABLE STATEMENT, THUS THE EXECUTION OF A NEW OPTION STATEMENT MAY REDEFINE PREVIOUSLY DEFINED OPTIONS. THE AVAILABLE OPTIONS OR TRANSFORMATIONS ARE AS FOLLOWS:

INT/NOINT

'INT' OPTION CAUSES THE EVALUATION OF THE INTEGER VALUED FUNCTIONS WHENEVER THEIR ARGUMENTS ARE INTEGER CONSTANTS. 'NOINT' OPTION LEAVES THESE FUNCTIONS UNEVALUATED.

MFCT/NOMFCT

'MFCT' OPTION CAUSES THE EVALUATION OF THE MATHEMATICAL FUNCTIONS WHENEVER THEIR ARGUMENTS ARE CONSTANTS. 'NOMFCT' OPTION LEAVES THESE FUNCTIONS UNEVALUATED.

PRODEX/NOPREX

'PRODEX' CAUSES THE EXPANSION OF EXPONENTIALIZATIONS OVER PRODUCTS, I.E. (E1*E2*...*EN)**F => (E1**F)*...*(EN**F). NOPREX TURNS THE OPTION OFF, I.E. THE ABOVE TRANSFORMATION IS NOT PERFORMED.

EXPAND(N)

WHERE 'N' MUST BE A NON-NEGATIVE INTEGER. THIS OPTION ALLOWS OR PROHIBITS THE AUTOMATIC USAGES OF THE DISTRIBUTIVITY AND/OR BINOMIAL EXPANSION.
N=1: THE RIGHT AND LEFT DISTRIBUTIVE LAW WILL BE APPLIED. I.E.

```
E1*(E2+E3) => E1*E2+E1*E3
(E1+E2)*E3 => E1*E3+E2*E3 ;
```

BUT THIS LEAVES THE EXPRESSIONS OF THE FORM (E1+E2+...+EN)**K WHERE K IS AN INTEGER, UNCHANGED, I.E. THE BINOMIAL LAW IS NOT APPLIED.
N>1: IN ADDITION TO THE USE OF THE DISTRIBUTIVE LAW, THE EXPONENTIAL EXPRESSIONS ARE ALSO EXPANDED ACCORDING TO THE BINOMIAL EXPANSION WHENEVER K IS LESS THAN OR EQUAL TO N.
N=0: THIS PROHIBITS THE USE OF THE DISTRIBUTIVE AND BINOMIAL LAWS.

BASE(K)

WHERE 'K' IS EITHER LETTER 'E' OR INTEGER 0,2 OR 10.
K=0: THIS LEAVES ALL CONSTANTS RAISED TO A NON-CONSTANT ARGUMENT (E.G. C**E) UNCHANGED.
K=2 OR 10: THIS CAUSES THE CONSTANT BASE IN C**E TO BE TRANSFORMED TO 2 OR 10 , RESPECTIVELY. I.E.

```
C**E => 2**(E*LOG[C]/LOG[2])        K=2
C**E => 10**(E*LOG[C]/LOG[10])      K=10
```

K=E: THIS CAUSES THE TRANSFORMATION OF THE ABOVE EXPRESSION INTO EXPONENTIAL FUNCTION, I.E.

```
C**E => EXP[E*LOG[C]]
```

EXAMPLE:

```
OPTION EXPAND(10),NOINT
```

3.2.12. ERASE STATEMENT

THE ERASE STATEMENT ATOMIZES SPECIFIED VARIABLES, I.E. RESETS THEIR VALUES TO THEIR NAMES. THIS STATEMENT FREES THE STORAGE LOCATION OCCUPIED BY THE EXPRESSION VALUES OF THE LISTED VARIABLES. THE USE OF THIS STATEMENT MAY BE ESSENTIAL TO AVOID EXHAUSTING FREE STORAGE DURING THE EXECUTION OF A LARGER FORMAL PROGRAM. THE FORM OF THE STATEMENT IS AS FOLLOWS:

```
ERASE  V1,V2,...,VN
```

WHERE V1,...,VN ARE NAMES OF F-VARIABLES. VI MAY APPEAR WITH OR WITHOUT SUBSCRIPT. WHEN VI IS LISTED WITH A SUBSCRIPT, E.G. A(1,2), THEN ONLY THE VARIABLE WITH THE SPECIFIED SUBSCRIPT IS RESET. WHEN VI IS LISTED WITHOUT A SUBSCRIPT, E.G. A, AND VI WAS USED WITH SUBSCRIPTS PREVIOUSLY, THEN ALL VARIABLES WITH THE SAME NAME AND DIFFERENT SUBSCRIPTS ARE RESET. EXAMPLE:

```
ERASE  A(1,1),A(2,1),B,X
```

3.2.13. DUMP STATEMENT

THE FOLLOWING 4 VARIATIONS OF DUMP STATEMENT ARE AVAILABLE:

```
DUMP SYMBOLS
DUMP EXPRESSIONS
DUMP ALL SYMBOLS
DUMP ALL EXPRESSIONS
```

DUMPING SYMBOLS GIVES A LIST OF THE F-VARIABLES WHICH HAD BEEN ASSIGNED PREVIOUSLY. DUMPING EXPRESSIONS GIVES THE LIST OF ASSIGNED F-VARIABLES TOGETHER WITH THEIR EXPRESSION VALUES. THE ABSENCE OF 'ALL' MEANS THAT THE LIST IS RESTRICTED TO THE SUBPROGRAM WHICH CONTAINS THE DUMP STATEMENT. 'ALL' MEANS THAT THE LIST INCLUDES ALL VARIABLES/EXPRESSIONS FROM EVERY SUBPROGRAM.

3.2.14. ROLOUT STATEMENT

THE ROLOUT STATEMENT PROVIDES THE USER TO USE THE AVAILABLE CORE STORAGE THE MOST ECONOMICALLY. DURING EXECUTION, THE FORMAL SYMBOL TABLE AND SYMBOLIC EXPRESSIONS ARE STORED IN A RESERVED CORE AREA. WHEN THIS AREA IS EXHAUSTED, SYMBOLIC EXPRESSIONS ARE COPIED OUT TO THE DRUM AND THEIR PLACES ARE MADE AVAILABLE. THIS PROCESS IS AUTOMATIC, AND IT DOES NOT CHECK FOR THE TYPE OF EXPRESSIONS TO BE ROLLED OUT TO DRUM; THUS IT MAY HAPPEN THAT THE ROLLED OUT EXPRESSION IS NEEDED A COUPLE OF STEPS LATER. THE USER MAY AVOID THIS DISADVANTAGE BY SPECIFYING THE ASSIGNED VARIABLES, V1,V2,...,VN, IN THE ROLOUT STATEMENT

```
ROLOUT V1,V2,...,VN
```

WHOSE EXPRESSION VALUES ARE NOT USED IN THE NEAR FUTURE, THUS THEY CAN BE ROLLED OUT TO PROVIDE MORE SPACE IN CORE.

3.2.15. SAVE AND RESET STATEMENTS

THE SAVE STATEMENT PROVIDES THE USER TO SAVE THE RESULTS OF HIS SYMBOLIC CALCULATIONS, I.E. THE OBJECT TIME FORMAL SYMBOL TABLE AND SYMBOLIC EXPRESSIONS, IN AN EXTERNAL FILE. THE RESET STATEMENT PROVIDES THE RETRIEVAL OF A PREVIOUSLY SAVED RESULTS. IT IS IMPORTANT TO KNOW THAT THE ENTIRE FORMAL OBJECT TIME DATA STORAGE IS SAVED, WHICH OVERWRITES THE DATA STORAGE WHEN THE RESET STATEMENT IS EXECUTED. FURTHERMORE, SINCE THE FORMAL DATA STRUCTURE CONTAINS THE USER'S SUBPROGRAM NAMES (FIRST 6 CHARACTERS OF THE ELEMENT NAME) IN ITS SYMBOL TABLE, THE RESET STATEMENT MUST APPEAR EITHER IN THE SAME RUN, OR IT HAS TO HAVE SUBPROGRAMS WITH NAMES CORRESPONDING TO THE NAMES OF SUBPROGRAMS WHERE THE SAVE STATEMENT OCCURED.

THE FORMAT OF THE SAVE AND RESET STATEMENTS ARE AS FOLLOWS:

```
SAVE  FILE.ELT/VERSION
RESET FILE.ELT/VERSION
```

OR

```
SAVE  .ELT/VERSION
RESET .ELT/VERSION
```

WHERE FILE IS THE NAME OF THE EXTERNAL FILE AND IT MUST BE ASSIGNED BY THE USER PRIOR TO THE EXECUTION OF HIS PROGRAM. WHEN FILE IS NOT SPECIFIED, THEN IT IS ASSUMED TO BE TPF$. ELT/VERSION IS THE NAME AND VERSION OF THE ELEMENT IN THE FILE. VERSION IS OPTIONAL, AND IT MAY BE DELETED.

3.3. FORTRAN-VALUED FUNCTIONS

THE FORTRAN VALUED FUNCTIONS ARE FUNCTIONS AVAILABLE FOR USE IN FORTRAN STATEMENTS. THEIR ARGUMENTS ARE F-EXPRESSIONS ENCLOSED IN APOSTROPHES. THE AVAILABLE FUNCTIONS ARE AS FOLLOWS:

```
LDOP('E')
```

GIVES AN INTEGER CORRESPONDING TO THE LEAD OPERATOR OF THE F-EXPRESSION E. THE CORRESPONDENCE IS AS FOLLOWS:

CONSTANT ZERO	0	EXP	32
INTEGER CONSTANT	1	LOG	33
RATIONAL CONSTANT	2	TNH	34
REAL CONSTANT	3	SIN	35
UNDEFINED FUNCTION ARGUMENT	5	COS	36
		ATN	37
NON-SUBSCR. VARIABLE	6	FAC	50
SUBSCRIPTED VARIABLE	7	BIN	51
COMMA (LIST)	16	STEP	52
+ (SUM)	18		
* (PRODUCT)	19		
** (EXPONENTIALIZATION)	20		
DIFF. OPERATOR	23		
UNDEFINED FUNCTION	25-31		

EXAMPLE: I = LDOP('X+Y+1')

WITH ATOMIC F-VARIABLE, X AND Y, THE EXECUTION OF THIS FORTRAN STATEMENT SETS I EQUAL TO 18 CORRESPONDING TO A SUM.

NARG('E')

GIVES THE NUMBER OF ARGUMENTS OF THE LEAD OPERATOR OF THE F-EXPRESSION, E. WHEN E IS A CONSTANT, VARIABLE, OR UNDEFINED FUNCTION ARGUMENT, THEN NARG GIVES ZERO.

EXAMPLE: J=NARG('X+Y+1')

SETS THE VALUE OF J TO 3.

IVALUE('E')

VALUE('E')

GIVE ZERO WHENEVER THE F-EXPRESSION E IS NOT A CONSTANT, OTHERWISE THEY GIVE THE CONSTANT VALUE EITHER IN INTEGER FORM (IVALUE) OR IN REAL FORM (VALUE).

IDENT('E1,E2')

GIVES LOGICAL VALUE .TRUE. WHEN THE TWO F-EXPRESSIONS ARE IDENTICAL; GIVES .FALSE. OTHERWISE.

EXAMPLE: 'OPTION EXPAND(0)'

```
I=IDENT('X*(X+Y),X**2+X*Y')

'OPTION EXPAND(5)'

J=IDENT('X*(X+Y),X**2+X*Y')
```

THE EXECUTION OF THE ABOVE STATEMENTS SETS I TO .FALSE. AND J TO .TRUE.

NCOUNT(0)

GIVES THE NUMBER OF AVAILABLE STORAGE LOCATIONS IN CORE FOR THE SYMBOLIC EXPRESSIONS. THE ARGUMENT 0 IS A DUMMY ARGUMENT.

3.4. FORMAL VARIABLES AS SUBPROGRAM ARGUMENTS

THE FORMAL SYSTEM ALLOWS FORMAL VARIABLES AS PART OF THE ARGUMENT LIST IN SUBPROGRAMS (FUNCTION OR SUBROUTINE) WITH SOME RESTRICTIONS. THE RULES TO BE FOLLOWED ARE AS FOLLOWS:

IN THE CALLING ROUTINE:

```
CALL SUB(...,'V;',...) OR
CALL SUB(...,'V(E);',...)
```

WHERE V MUST BE A VARIABLE OR FUNCTION NAME WHICH MAY BE SUBSCRIPTED BY E FOLLOWING THE RULES OF SUBSCRIPTION, BUT IT MAY NOT CONTAIN FORTRAN TYPE OF EXPRESSIONS.

THE ARGUMENT MUST BE TERMINATED BY SEMICOLON AND ENCLOSED IN APOSTROPHES.

IN THE CALLED ROUTINE:

```
SUBROUTINE SUB(...,'A',...)
```

THE CORRESPONDING ARGUMENT NAME MUST BE ENCLOSED IN APOSTROPHES, AND IT MUST APPEAR WITHOUT SUBSCRIPT.

EXAMPLE:

```
CALL SUB('A;')          SUBROUTINE SUB('B')
                        B=2*X

CALL SUBR('C;')         SUBROUTINE SUBR('D')
                        D(2,3)=X**2
```

3.5. SUMMARY OF RESTRICTIONS.

1. RESTRICTED EXTERNAL NAMES.
   NAMES STARTING WITH 'FML' ARE RESERVED FOR FORMAL OBJECT TIME ROUTINES.

2. MAXIMUM NUMBER OF ARGUMENTS IN FORMAL FUNCTIONS.

DEFINED FUNCTION	20
UNDEFINED FUNCTION	7
SUBST	1+2*20
REPLAC	1+2*20

3. SUBSCRIPTS.
   MAXMUM 4
   INTEGER VALUED IN RANGE 0 - 511

4. DIMENSIONALITY OF F-VARIABLES.
   DIMENSIONALITY OF AN F-VARIABLE, I.E. SUBSCRIPTED OR NOT, IS DEFINED BY ITS FIRST USE ON THE LEFT SIDE OF AN ASSIGN STATEMENT. SUBSEQUENT USE OF THE VARIABLE MUST CORRESPOND TO IT, I.E. A NON-SUBSCRIPTED VARIABLE MUST BE USED WITHOUT SUBSCRIPT, A SUBSCRIPTED VARIABLE MUST BE USED WITH SUBSCRIPT(S). THERE IS NO RESTRICTION ON THE USE OF A SUBSCRIPTED VARIABLE WITH DIFFERENT NUMBERS OF SUBSCRIPTS, E.G. A(5) AND A(2,0,6) MAY BE USED IN THE SAME PROGRAM.
   WHEN AN F-VARIABLE WHICH DOES NOT APPEAR AS SUBPROGRAM ARGUMENT IS ERASED BY AN ERASE STATEMENT, THEN ITS DIMENSIONALITY MAY BE REDEFINED.

5. SUBPROGRAM ARGUMENTS.
   ONLY F-VARIABLES WITHOUT SUBSCRIPTS AND DEFINED FUNCTIONS MAY BE USED AS ARGUMENTS.
   ERASING THE VALUE OF AN F-VARIABLE WHICH IS A SUBPROGRAM ARGUMENT RESETS THE VALUE TO ITSELF BUT RETAINS THE CORRESPONDENCE OF THE ARGUMENT TO THE CALLING AND CALLED ARGUMENTS.

4. EXAMPLES OF THE USE OF THE FORMAL SYSTEM

4.1. TAYLOR EXPANSION

```
@FORMAL*LIB.FORMAL,IL       ELT1,ELT2
C      THIS PROGRAM GENERATES A TAYLOR EXPANSION IN THE
C      NEIGHBORHOOD OF A, FOR A GIVEN FUNCTION FX, WHICH
C      HAS CONTINUOUS DERIVATIVES UP TO N-TH ORDER
C
       LOGICAL IDENT
       'OPTION INT, EXPAND(12),MFCT'
     1 'READ N,A,FX'
       N = IVALUE('N')
       A = IVALUE('A')
       PRINT 8
       'PRINT FX'
       PRINT 9,N,A
       'F=REPLAC[FX,X,A]'
       DO  2 I=1,N
       'I = #I(I)'
       'FX = DIF[FX,X,1]'
       IF (IDENT('FX,0')) STOP
       'FF = FIX[REPLAC[FX,X,A],-4]'
     2 'F=F+1/FAC[I]*(X-A)**I*FF'
       'PRINT F'
       PRINT '(/////)'
       GO TO 1
     8 FORMAT (' THE TAYLOR SERIES EXPANSION FOR THE FUNCTION')
     9 FORMAT (' OF ORDER',I3,', IN THE NEIGHBORHOOD OF X =',I3)
       END
@MAP
 LIB  FORMAL*LIB.
@XQT
 10 ; 0 ; SIN[X] ;
 10 ; 0 ; COS[X] ;
 10 ; 0 ; C*(1-2*C**2*X)**(-1/2) ;
```

```
CYCLE 00: 09 SEP 70 AT 11:29:22 FORMAL 03:02

C       THIS PROGRAM GENERATES A TAYLOR EXPANSION IN THE
C       NEIGHBORHOOD OF A, FOR A GIVEN FUNCTION FX, WHICH
C       HAS CONTINUOUS DERIVATIVES UP TO N-TH ORDER
        LOGICAL IDENT
        'OPTION INT, EXPAND(12),MFCT'
      1 'READ N,A,FX'
        N = IVALUE('N')
        A = IVALUE('A')
        PRINT 8
        'PRINT FX'
        PRINT 9,N,A
        'F=REPLAC[FX,X,A]'
        DO  2 I=1,N
        'I = #I(I)'
        'FX = DIF[FX,X,1]'
        IF (IDENT('FX,0')) STOP
        'FF = FIX[REPLAC[FX,X,A],-4]'
      2 'F=F+1/FAC[I]*(X-A)**I*FF'
        'PRINT F'
        PRINT '(/////)'
        GO TO 1
      8 FORMAT ('1THE TAYLOR SERIES EXPANSION FOR THE FUNCTION')
      9 FORMAT (' OF ORDER',I3,'IN THE NEIGHBORHOOD OF X =',I3)
        END
```

PREPROCESSOR OUTPUT :

```
      LOGICAL IDENT
      CALL FMLOPT('INT,EXPAND(12),MFCT;',0)
    1 CALL FMLIO1('N,A,FX;',0)
      N = IVALUE('N;',0)
      A = IVALUE('A;',0)
      PRINT 8
      CALL FMLIO2('FX;',0)
      PRINT 9,N,A
      CALL FMLASG ('F=REPLAC[FX,X,A];',0)
      DO  2 I=1,N
      CALL FMLASG ('I=#I1;',1,I)
      CALL FMLASG ('FX=DIF[FX,X,1];',0)
      IF (IDENT('FX,0;',0)) STOP
      CALL FMLASG ('FF=FIX[REPLAC[FX,X,A],-4];',0)
    2 CALL FMLASG ('F=F+I/FAC[I]*(X-A)**I*FF;',0)
      CALL FMLIO2('F;',0)
      PRINT '(/////)'
      GO TO 1
    8 FORMAT ('1THE TAYLOR SERIES EXPANSION FOR THE FUNCTION')
    9 FORMAT (' OF ORDER',I3,'IN THE NEIGHBORHOOD OF X =',I3)
      END
```

```
FORMALLIB VERSION 07:09A

 THE TAYLOR SERIES EXPANSION FOR THE FUNCTION
 FX=
 SIN[X]
 OF ORDER 10, IN THE NEIGHBORHOOD OF X =  0
 F=
 X - 1/6 * X ** 3 + 1/120 * X ** 5 - 1/5040 * X ** 7 + 1/362880 *
 X ** 9



 THE TAYLOR SERIES EXPANSION FOR THE FUNCTION
 FX=
 COS[X]
 OF ORDER 10, IN THE NEIGHBORHOOD OF X =  0
 F=
 1.00000 - 1/2 * X ** 2 + 1/24 * X ** 4 - 1/720 * X ** 6 +
 1/40320 * X ** 8 - 1/3628800 * X ** 10



 THE TAYLOR SERIES EXPANSION FOR THE FUNCTION
 FX=
 C / (1 - 2 * C** 2 * X)  ** 1/2
 OF ORDER 10, IN THE NEIGHBORHOOD OF X =  0
 F=
 C + C** 3 * X + 3/2 * C ** 5 * X ** 2 + 5/2 * C ** 7 * X ** 3 +
 35/8 * C ** 9 * X ** 4 + 63/8 * C ** 11 * X ** 5 + 231/16 * C **
 13 * X ** 6 + 429/16 * C ** 15 * X ** 7 + 6435/128 * C ** 17 * X
 ** 8 + 12155/128 * C ** 19 * X ** 9 + 46189/256 * C ** 21 * X **
 10
```

4.2. FACTORING

```
@FORMAL*LIB.FORMAL,IL    ELT1,ELT2
C   **** THIS IS THE DUMMY MAIN ROUTINE FOR TESTING THE
C   **** FACTORING SUBROUTINE.
 1    'READ E'
      PRINT '('' THE ORIGINAL EXPRESSION IS '')'
      'PRINT E'
      CALL FACTOR ('E;')
      PRINT '('' THE RESULT AFTER FACTORING IS :'')'
      'PRINT E'
      PRINT '(/////)'
      GO TO 1
      END

@FORMAL*LIB.FORMAL,IL   ELT3,ELT4
C         THIS ROUTINE FACTORS OUT THE COMMON FACTOR IN AN
C         EXPRESSION WHICH IS A SUM OF PRODUCTS.
C         IF EXPRESSION 'E' IS NOT A SUM, THEN THE EXPRESSION
C         'E' WILL BE RETURNED WITHOUT FACTORING.
C         E = A1+A2+A3+...+AN = F*(B1+B2+...+BN)
C         THE CONSTANT FACTOR WILL BE DELETED
      SUBROUTINE FACTOR ('E')
      'OPTION PRODEX,EXPAND(0)'
C    ***** IF THE LEAD OPERATOR IS NOT '+', THEN RETURN
      IF (LDOP('E') .NE. 18) RETURN
      'E = EXPAND[E]'
      'FC = (NUM[[1]]/NUM[NUM[[1]]/NUM[[2]]])/(DENOM[[1]]/
     + NUM[DENOM[[1]]/DENOM[[2]]])'
      N = NARG('E')
C    ***** TAKE THE FIRST ARGUMENT AS ITS FACTOR
      'F = ARG[E,1]'
C    ***** LOOP TO COMPUTE THE COMMON FACTOR OF EXPRESSION E
      DO 50 I=2,N
      'A = ARG[E,#I(I)]'
C    ***** USE THE DEFINED FUNCTION 'FC'
 50   'F = FC[F,A]'
      'FF = 0'
C    ***** DELETE THE CONSTANT FACTOR
      'BB = F'
      IF (LDOP('BB') .EQ. 19) 'BB=ARG[BB,1]'
      IF (LDOP('BB') .GT. 3) GO TO 100
      'F = F/BB'
C    ***** LOOP TO THE REMAINING PART OF THE EXPRESSION
 100  DO 150 I=1,N
 150  'FF = FF+(ARG[E,#I(I)]/F)'
      'E = F*FF'
C    ***** RETURN TO THE CALLING PROGRAM
      RETURN
```

```
      END
@MAP
 LIB   FORMAL*LIB.
@XQT
 A1*X+A2*X+A3*(A1+A2)*X**2 ;
 2/3*X**2+3/4*X*Y**2+4*X**3+1/2*X*Y ;
 1/2*(X+Y) ;
```

```
CYCLE 00 : 09 SEP 70 AT 10:30:22 FORMAL 03:02

C    **** THIS IS THE DUMMY MAIN ROUTINE FOR TESTING THE
C    **** FACTORING SUBROUTINE.
 1     'READ E'
       PRINT '('' THE ORIGINAL EXPRESSION IS '')'
       'PRINT E'
       CALL FACTOR ('E;')
       PRINT '('' THE RESULT AFTER FACTORING IS :'')'
       'PRINT E'    .
       PRINT '(/////)'
       GO TO 1
       END


         PREPROCESSOR OUTPUT :
 1     CALL FMLIO1('E;',0)
       PRINT '('' THE ORIGINAL EXPRESSION IS '')'
       CALL FMLIO2('E;',0)
       CALL FACTOR ('E;')
       PRINT '('' THE RESULT AFTER FACTORING IS :'')'
       CALL FMLIO2('E;',0)
       PRINT '(/////)'
       GO TO 1
       END
```

```
CYCLE 00 : 09 SEP 70 AT 10:31:45 FORMAL 03:02

C         THIS ROUTINE FACTORS OUT THE COMMON FACTOR IN AN
C         EXPRESSION WHICH IS A SUM OF PRODUCTS.
C         IF EXPRESSION 'E' IS NOT A SUM, THEN THE EXPRESSION
C         'E' WILL BE RETURNED WITHOUT FACTORING.
C         E = A1+A2+A3+...+AN = F*(B1+B2+...+BN)
C         THE CONSTANT FACTOR WILL BE DELETED
       SUBROUTINE FACTOR ('F')
       'OPTION PRODEX,EXPAND(0)'
C     ***** IF THE LEAD OPERATOR IS NOT '+', THEN RETURN
       IF (LDOP('E') .NE. 18) RETURN
       'E = EXPAND[E]'
       'FC = (NUM[[1]]/NUM[NUM[[1]]/NUM[[2]]])/(DENOM[[1]]/
      + NUM[DENOM[[1]]/DENOM[[2]]])'
       N = NARG('E')
C     ***** TAKE THE FIRST ARGUMENT AS ITS FACTOR
       'F = ARG[E,1]'
C     ***** LOOP TO COMPUTE THE COMMON FACTOR OF EXPRESSION E
       DO 50 I=2,N
       'A = ARG[E,#I(I)]'
C     ***** USE THE DEFINED FUNCTION 'FC'
 50    'F = FC[F,A]'
       'FF = 0'
C     ***** DELETE THE CONSTANT FACTOR
       'BB = F'
       IF (LDOP('BB') .EQ. 19) 'BB=ARG[BB,1]'
       IF (LDOP('BB') .GT. 3) GO TO 100
       'F = F/BB'
C     ***** LOOP TO THE REMAINING PART OF THE EXPRESSION
 100   DO 150 I=1,N
 150   'FF = FF+(ARG[E,#I(I)]/F)'
       'E = F*FF'
C     ***** RETURN TO THE CALLING PROGRAM
       RETURN
       END
```

PREPROCESSOR OUTPUT :

```
      SUBROUTINE FACTOR (*)
      CALL FMLEQU(1,'E      ')
      CALL FMLOPT('PRODEX,EXPAND(0);',0)
      IF (LDOP('E;',0) .NE. 18) RETURN
      CALL FMLASG ('E=EXPAND[E];',0)
      CALL FMLASG ('FC=(NUM[[1]]/NUM[NUM[[1]]/NUM[[2]]])/(DENOM
     +[[1]]/NUM[DENOM[[1]]/DENOM[[2]]]);',0)
      N = NARG('E;',0)
      CALL FMLASG ('F=ARG[E,1];',0)
      DO 50 I=2,N
      CALL FMLASG ('A=ARG[E,#I1];',1,(I))
50    CALL FMLASG ('F=FC[F,A];',0)
      CALL FMLASG ('FF=0;',0)
      CALL FMLASG ('BB=F;',0)
      IF (LDOP('BB;',0) .EQ. 19) CALL FMLASG ('BB=ARG[BB,1];',0)
      IF (LDOP('BB;',0) .GT. 3) GO TO 100
      CALL FMLASG ('F=F/BB;',0)
100   DO 150 I=1,N
150   CALL FMLASG ('FF=FF+(ARG[E,#I1]/F);',1,(I))
      CALL FMLASG ('E=F*FF;',0)
      RETURN
      END
```

```
FORMALLIB VERSION 07:09A

 THE ORIGINAL EXPRESSION IS
 E=
 A1 * X + A2 * X + A3 * X ** 2 * (A1 + A2)

 THE RESULT AFTER FACTORING IS :
 E=
 X * (A1 + A2 + A1 * A3 * X + A2 * A3 * X)



 THE ORIGINAL EXPRESSION IS
 E=
 1/2 * X * Y + 3/4 * X * Y ** 2 + 2/3 * X ** 2 + 4 * X ** 3

 THE RESULT AFTER FACTORING IS :
 E=
 X * (2/3 * X + 1/2 * Y + 4 * X ** 2 + 3/4 * Y ** 2)



 THE ORIGINAL EXPRESSION IS
 E=
 1/2 * ( X + Y)

 THE RESULT AFTER FACTORING IS :
 E=
 1/2 * (X + Y)
```

5. INTERACTIVE FORMAL SYSTEM

A SUBSET OF THE FORMAL LANGUAGE IS AVAILABLE AS AN INTERACTIVE SYSTEM FROM TELETYPE. IN THIS SUBSET, THERE ARE NO FORTRAN TYPE STATEMENTS AVAILABLE, THUS FORTRAN-TYPE CONSTANTS IN FORMAL STATEMENTS ARE ALSO NOT AVAILABLE. THE AVAILABLE STATEMENTS ARE AS FOLLOWS:

OPTION
ERASE
ASSIGN
PRINT
COMMENT
NCOUNT
ROLOUT
SAVE
RESET

THESE STATEMENTS SHOULD * N O T * BE ENCLOSED IN APOSTROPHES CONTRARY TO THE FORMAL SYSTEM DESCRIBED IN CHAPTER 3. THE TYPE OF THE STATEMENT IS DETERMINED EITHER BY THE FIRST 2 OR THE FIRST 6 CHARACTERS TYPED ON A LINE. WITH Δ INDICATING A SPACE, THESE FORMS ARE AS FOLLOWS:

CΔ	COMMENT STATEMENT
COMMEN	COMMENT STATEMENT
PΔ	PRINT STATEMENT
PRINTΔ	PRINT STATEMENT
OPTION	OPTION STATEMENT
ERASEΔ	ERASE STATEMENT
NCOUNT	IT GIVES THE NUMBER OF AVAILABLE CORE LOCATIONS
ROLOUT	
SAVEΔΔ	SAVE STATEMENT
RESETΔ	RESET STATEMENT

WHENEVER A TYPED-IN LINE DOES NOT HAVE ONE OF THE ABOVE FORMS, THE SYSTEM AUTOMATICALLY ASSUMES THAT IT IS AN ASSIGN STATEMENT. ERROR MESSAGES ARE GIVEN WHEN THE STATEMENTS ARE SYNTACTICALLY INCORRECT.

AT THE PRESENT, THE STATEMENTS ARE RESTRICTED TO ONE STATEMENT PER LINE. WHEN A LINE IS TYPED, AND THE CARRIAGE RETURN IS PUSHED, THE LINE IS ACCEPTED AS A STATEMENT AND EXECUTED. BEFORE EXECUTION, THE STATEMENT IS REPRINTED TO PROVIDE A LISTING WITHOUT THE CORRECTION ARROWS WHICH MIGHT POSSIBLY BE EMPLOYED IN THE INPUT IMAGE. THE OPTION, ERASE, AND ASSIGN STATEMENTS ARE THE SAME SYNTACTICLY AND SEMANTICLY AS THE CORRESPONDING STATEMENTS IN CHAPTER 2. THE COMMENT STATEMENT CORRESPONDS TO THE FORTRAN COMMENT STATEMENT. THE PRINT STATEMENT CORRESPONDS TO THE FORMAL OUTPUT STATEMENT WITH FIXED FORMAT; IT MUST BE FOLLOWED BY A LIST OF VARIABLES, THE SYSTEM PRINTS OUT THE VARIABLES FOLLOWED BY EQUAL SIGNS AND THEIR EXPRESSION VALUES. THE VARIABLES AND THEIR EXPRESSIONS ALWAYS START ON A NEW LINE IN COLUMN 2. WHEN AN

EXPRESSION IS LONGER THAN ONE LINE, IT IS CONTINUED ON A NEW LINE STARTING IN COLUMN 2.

THE INTERACTIVE FORMAL SYSTEM CAN BE ACCESSED FROM THE SYSTEM BY

```
@RUN  ID,III-JJ-KKK,JOHN-DOE
@XQT  FORMAL*LIB.FML
```

EXEC 8 CONTROL LINES, AND ITS USE IS TERMINATED BY ANY EXEC 8 CONTROL LINE, I.E. @ IN COLUMN 1.

IF A SET OF FORMAL STATEMENTS MUST BE EXECUTED A NUMBER OF TIMES, THEN THE FOLLOWING PROCEDURES CAN BE UTILIZED TO SAVE A CONSIDERABLE AMOUNT OF TEDIOUS TYPING . FIRST, A SYMBOLIC ELEMENT CALLED 'LOOP' SHOULD BE CREATED BY '@ELT,S LOOP' WHICH IS THEN FOLLOWED BY THE SET OF FORMAL STATEMENTS. AFTER ALL STATEMENTS ARE TYPED IN, TERMINATE THE ELEMENT PROCESSOR BY TYPING IN MASTER CONTROL '@' IN THE FIRST COLUMN. SECOND, TO UTILIZE THIS 'LOOP' ELEMENT SIMPLY TYPE IN '@ADD LOOP' WHEREVER THIS SET OF INSTRUCTIONS IS NEEDED. SEE EXAMPLE 2 IN SECTION 6.2.

6. EXAMPLES OF THE INTERACTIVE FORMAL SYSTEM

6.1. EXAMPLE 1

```
@RUN  ID,000-00-000,JOE

@XQT  FORMAL*LIB.FML
FORMALLIB VERSION 05.05
C   THE ABOVE STATEMENT INDICATES WHICH VERSION IS WORKING
C   NOW!
C        THIS EXAMPLE USES FEATURES OF SOME FORMAL STATEMENTS
C        TO PROVE THE EQUATION FOR THE SUM OF GEOMETRIC SERIES.
C        I.E. SUM OF (X**(I-1)) = (1-X**N)/(1-X)  FOR I=1,...,N
C        LET 'SN' BE THE SUM AND TRY TO PROVE IT BY MATHEMATICAL
C        INDUCTION.  FIRST, TEST TO SEE IF THE EQUATION HOLDS
C        FOR N=1,(I.E. THE SUM 'SN'=1) IF IT IS SO, THEN ASSUME
C        THE EQUATION IS TRUE FOR ALL INTEGERS LESS THAN OR
C        EQUAL TO N, THEN PROCEED TO SHOW THAT THE EQUATION IS
C        TRUE FOR INTEGER N+1.  ACCORDING TO THE EQUATION AND
C        SUBSTITUTION OF N+1 FOR N,  'SUM1'=(1-Z**N+1)/(1-Z);
C        BY THE ASSUMPTION AND THE DEFINITION OF THE SUMMATION
C        WE KNOW THAT  'SUM1'=SN+X**N.  THE OBJECTIVE IS THEN
C        TO SHOW THAT 'SUM1' IS IDENTICAL TO 'SUM2'  BY TAKING
C        THEIR DIFFERENCE.
C
OPTION EXPAND(0)
 OPTION: EXPAND(0)
SN = (1-X**N)/(1-X)
 SN = (1-X**N)/(1-X)
TEST1 = SUBST[SN,N,1]
 TEST1 = SUBST[SN,N,1]
PRINT TEST1
 TEST1=
  1
C
SUM1 = SUBST[SN,N,N+1]
 SUM1 = SUBST[SN,N,N+1]
PRINT SUM1
 SUM1=
  (1 - 1 * X **  (1 + N) )  *  (1 - 1 * X)  ** ( -1)
SUM2 = SN+X**N
 SUM2 = SN+X**N
PRINT SUM2
 SUM2=
  (1 - 1 * X** N)  *  (1 - 1 * X)  ** ( -1)  + X ** N
```

```
PROOF = SUM1-SUM2
 PROOF = SUM1-SUM2
PRINT PROOF
 PROOF=
  - 1 *  ( (1 - 1 * X ** N)  *  (1 - 1 * X)  ** ( -1)  + X ** N)
 +  (1 - 1 * X **  (1 + N) )  *  (1 - 1 * X)  ** ( -1)
C
C        SINCE 'PROOF' IS NOT ZERO. LET'S PUT IT IN THE COMMON
C        DENOMINATOR FORM.
C
EXAM = EXPAND[PROOF]
 EXAM = EXPAND[PROOF]
PRINT EXAM
 EXAM=
  ( - 1 *  (1 - 1 * X)  * X ** N + X ** N - 1 * X **  (1 + N) )
 *  (1 - 1 * X)  ** ( -1)
ZERO = EXPAND[NUM[EXAM]]/DENOM[EXAM]
 ZERO = EXPAND[NUM[EXAM]]/DENOM[EXAM]
PRINT ZERO
 ZERO=
 0
```

6.2. EXAMPLE 2

THIS EXAMPLE SHOWS HOW TO GENERATE A SET OF ORTHOGONAL POLYNOMIALS BY ITS RECURSIVE DEFINITION :
P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1) WHERE THE VALUE OF P(N), A SUBSCRIPTED VARIABLE, IS THE N'TH DEGREE POLYNOMIAL IN X. A[N], B[N], C[N] AS FUNCTIONS WITH ARGUMENT N, CORRESPOND TO THE COEFFICIENTS IN THE RECURSIVE FORMULA. TO AVOID THE TEDIOUS WORK OF TYPING THE ABOVE FORMULA REPEATEDLY BY INCREMENTING N AND PRINTING OUT P(N), WE DEFINE AN ELEMENT NAMED 'LOOP' AS FOLLOWS::

```
@ELT,IS   LOOP
P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
N = N+1
PRINT P(N)
@
```

THIS WAY WE ARE ABLE TO USE
@ADD LOOP
INSTEAD OF TYPING IN THE ABOVE 3 FORMAL STATEMENTS.
* * * * * * * * * * * * * * * * * * * *
NOW WE CALL 'FORMAL*LIB.FML' TO GENERATE THE LEGENDRE POLYNOMIALS.

```
@XQT   FORMAL*LIB.FML
FORMALLIB VERSION 05.05
OPTION EXPAND(10)
 OPTION: EXPAND(10)
C
C        FIRST INITIALIZE P(0),P(1)
C
P(0) = 1
 P(0) = 1
P(1) = X
 P(1) = X
C
C        NOW DEFINE THE FUNCTION A,B,AND C FOR LEGENDRE POLY-
C        NOMIAL. USE  '@ADD  LOOP'  REPEATEDLY TO OBTAIN THE
C        CONSECUTIVE POLYNOMIALS.
C
A = (2*[1]+1)/([1]+1)
 A = (2*[1]+1)/([1]+1)
B = 0
 B = 0
 C = [1]/([1]+1)
  C = [1]/([1]+1)
N = 1
 N = 1
```

```
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (2) =
   1/2 + 3/2 * X ** 2
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (3) =
   3/2 * X + 5/2 * X ** 3
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (4) =
 3/8 - 15/4 * X ** 2 + 35/8 * X ** 4
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (5) =
 15/8 * X - 35/4 * X ** 3 + 63/8 * X ** 5
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (6) =
   5/16 + 105/16 * X ** 2 - 315/16 * X ** 4 + 231/16 * X ** 6
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (7) =
   35/16 * X + 315/16 * X ** 3 - 693/16 * X ** 5 + 429/16 * X **
 7
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (8) =
 35/128 - 315/32 * X ** 2 + 3465/64 * X ** 4 - 3003/32 * X ** 6 +
 6435/128 * X ** 8
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (9) =
 315/128 * X - 1155/32 * X ** 3 + 9009/64 * X ** 5 - 6435/32 * X
 ** 7 + 12155/128 * X ** 9
@ADD     LOOP
 P(N+1) = (A[N]*X+B[N])*P(N)-C[N]*P(N-1)
 N = N+1
 P (10) =
   63/256 + 3465/256 * X ** 2 - 15015/128 * X ** 4 + 45045/128 *
 X ** 6 - 109395/256 * X ** 8 + 46189/256 * X ** 10
```

6.3. EXAMPLE 3

THIS EXAMPLE TAKEN FROM HOME WORK PROBLEMS OF FRESHMEN CALCULUS COURSE, SHOWS THE SIMPLICITY OF THE FORMAL PROGRAM IN SOLVING PARTIAL DERIVATIVES AND IN EVALUATING THEM AT A GIVEN POINT. THIS EXAMPLE PERFORMS SEVERAL SUBTASKS. ONE OF THESE IS TO PROVE ΔT**2 / ΔPΔA = ΔT**2 / ΔAΔP .

```
@XQT   FORMAL*LIB.FML
FORMALLIB VERSION 05.07
OPTION EXPAND(10)
 OPTION: EXPAND(10)
X = P*COS[A]
 X = P*COS[A]
Y = P*SIN[A]
 Y = P*SIN[A]
T = X**3-X*Y+Y**3
 T = X**3-X*Y+Y**3
DP = DIF[T,P,1]
 DP = DIF[T,P,1]
PRINT DP
 DP=
 3 *  (P * SIN [A] )  ** 2 * SIN [A]  + 3 *  (P * COS [A] )  ** 2
 * COS [A]  - 2 * P * SIN [A]  * COS [A]
C
V1 = SUBST[DP,A,3.1416/4]
 V1 = SUBST[DP,A,3.1416/4]
PRINT V1
 V1=
  - 1.00000 * P + 2.12132 *  (.70711 * P)  ** 2 + 2.12132 *
 (.70711 * P)  ** 2
C
DPA = DIF[DP,A,1]
 DPA = DIF[DP,A,1]
PRINT DPA
 DPA=
 2 * P * SIN [A]  ** 2 - 2 * P * COS [A]  ** 2 + 3 *  (P * SIN
 [A] )  ** 2 * COS [A]  - 3 *  (P * COS [A] )  ** 2 * SIN [A]  +
 6 * P ** 2 * SIN [A]  ** 2 * COS [A]  - 6 * P ** 2 * COS [A]  **
 2 * SIN [A]
C
DA = DIF[T,A,1]
 DA = DIF[T,A,1]
PRINT DA
 DA=
 P ** 2 * SIN [A]  ** 2 - 1 * P ** 2 * COS [A]  ** 2 + 3 * P *
 (P * SIN [A] )  ** 2 * COS [A]  - 3 * P *  (P * COS [A] )  ** 2
 * SIN [A]
C
```

```
V2 = SUBST[DA,A,3.1416/4]
 V2 = SUBST[DA,A,3.1416/4]
PRINT V2
 V2=
  - 2.12132 * P *  (.70711 *P)  ** 2 + 2.12132 * P *  (.70711 *
 P)  ** 2 + .00000 * P ** 2
C
DAP = DIF[DA,P,1]
 DAP = DIF[DA,P,1]
PRINT DAP
 DAP=
 2 * P * SIN [A]  ** 2 - 2 * P * COS [A]  ** 2 + 3 *  (P * SIN
 [A] )  ** 2 * COS [A]  - 3 *  (P * COS [A] )  ** 2 * SIN [A]  +
 6 * P ** 2 * SIN [A]  ** 2 * COS [A]  - 6 * P ** 2 * COS [A]  **
 2 * SIN [A]
C
V3 = SUBST[DPA,A,3.1416/4]
 V3 = SUBST[DPA,A,3.1416/4]
PRINT V3
 V3=
 .00001 * P + .00001 * P **2 - 2.12132 *  (.70711 * P)  ** 2 +
 2.12132 *  (.70711 * P)  ** 2
C
EQUAL = DPA-DAP
 EQUAL = DPA-DAP
PRINT EQUAL
 EQUAL=
 0
C
DP3 = DIF[T,P,3]
 DP3 = DIF[T,P,3]
DP3DA3 = DIF[DP3,A,3]
 DP3DA3 = DIF[DP3,A,3]
PRINT DP3DA3
 DP3DA3=
  - 126 * SIN [A]  ** 2 COS [A]  + 126 * COS [A]  ** 2 * SIN
 [A]  - 36 * SIN [A]  ** 3 + 36 * COS [A]  ** 3
C
V4 = SUBST[DP3DA3,A,3.1416/4]
 V4 = SUBST[DP3DA3,A,3.1416/4]
PRINT V4
 V4=
  - .00030
```

7. ERROR MESSAGES

ANY ERROR OCCURING DURING THE EXCUTION OF A FORMAL PROGRAM TERMINATES THE EXCUTION BY PRINTING OUT AN APPROPRIATE ERROR MESSAGE WITH ONE OF THE FOLLOWING FORMS:

E1 -- 'MESSAGE'
E2 -- 'MESSAGE'
E3 -- 'MESSAGE'

E1, E2, E3 CORRESPOND TO THREE TYPES OF ERRORS INDICATING THE NECESSARY CORRECTIONS TO BE MADE, TOGETHER WITH THE DIFFERENT MESSAGES.

E1 : THE STATEMENT CAUSING THE ERROR MUST BE CORRECTED. IT USUALLY CORRESPONDS TO A SYNTACTIC ERROR, AS STATED IN THE MESSAGE. IN THE INTERACTIVE FORMAL SYSTEM, IT CAN BE CORRECTED BY TYPING THE CORECT STATEMENT.

EXAMPLE : A = SUBST[X+Y,X,Y)
THIS STATEMENT CAUSES THE FOLLOWING ERROR MESSAGE TO BE PRINTED.

E1 -- DOES NOT MATCH THE LEFT SIDE)

E2 : THE PROCESS OF THE LAST STATEMENT EXHAUSTED THE AVAILABLE WORKING STORAGE. IT CAN BE CORRECTED BY USING A FORMAL 'ERASE' STATEMENT PRIOR TO THE LAST INDICATED STATEMENT. IN THE INTERACTIVE FORMAL SYSTEM, IT CAN BE CORRECTED BY TYPING IN AN 'ERASE' STATEMENT WITH APPROPRIATE PARAMETERS BEFORE TYPING IN THE LAST STATEMENT.

E3 : THE EXECUTION OF THE LAST STATEMENT CAUSES AN UNRECOVERABLE ERROR. IN THIS CASE THE PROGRAM USUALLY CAN NOT EASILY BE CORRECTED.

8. SYNTAX OF FORMAL STATEMENTS

IN THIS CHAPTER, THE SYNTACTIC RULES OF THE FORMAL STATEMENTS ARE PRESENTED IN BACKUS NORMAL FORM. THE LIMITATION OF THIS FORM REQUIRES SOME EXPLANATIONS WHICH ARE GIVEN SEMANTICALLY. THE FOLLOWING SYNTACTIC VARIABLES ARE USED WITHOUT FURTHER EXPLANATION:

<ID>::= IDENTIFIER CONSISTING OF NOT MORE THAN 6 ALPHANUMERIC CHARACTERS OF WHICH THE FIRST ONE IS A LETTER.

<I>::= DECIMAL INTEGER

<R>::= DECIMAL REAL NUMBER

<FNINT>::= FORTRAN ARITHMETIC EXPRESSION GIVING AN INTEGER

<FNREAL>::= FORTRAN ARITHMETIC EXPRESSION GIVING A REAL NUMBER

8.1.1. FORMAL CONSTANTS AND VARIABLES.

<F-CONSTANT>::= <F-INT> \ <F-REAL>

<F-INT>::= <I> \ #I(<FNINT>)

<F-REAL>::= <R> \ #R(<FNREAL>)

<F-VBLE>::= <F-SIMPLE-V> \ <F-SUBSC-V>

<F-SIMPLE-V>::= <ID>

<F-SUBSC-V>::= <ID>(<SUBS.LIST>)

<SUBS.LIST>::= <L.I> \ <L.I>,<L.I> \ <L.I>,<L.I>,<L.I> \ <L.I>,<L.I>,<L.I>,<L.I>

<L.I>::= <F-INT> \ <F-EXPR> SUCH THAT ITS VALUE IS AN INTEGER BETWEEN 0 AND 511.

8.1.2. FORMAL EXPRESSIONS.

ALGEBRAIC EXPRESSIONS

<F-EXPR>::= <F-SUM> \ +<F-SUM> \ -<F-SUM>

<F-SUM>::= <F-TERM> \ <F-SUM> + <F-TERM> \
<F-SUM> - <F-TERM>

<F-TERM>::= <F-FACTOR> \ <F-TERM>*<F-FACTOR> \
<F-TERM>/<F-FACTOR>

<F-FACTOR>::= <F-PRIME> \ <F-FACTOR>**<F-PRIME>

<F-PRIME>::= <F-CONSTANT> \ <F-VBLE> \ <F-FCT> \
(<F-EXPR>)

FUNCTION DEFINING EXPRESSIONS

<F-D-EXPR>::= <F-D-SUM> \ +<F-D-SUM> \ -<F-D-SUM>

<F-D-SUM>::= <F-D-TERM> \ <F-D-SUM>+<F-D-TERM> \
<F-D-SUM>-<F-D-TERM>

<F-D-TERM>::= <F-D-FACTOR> \ <F-D-TERM>*<F-D-FACTOR> \
<F-D-TERM>/<F-D-FACTOR>

<F-D-FACTOR>::= <F-D-PRIME> \ <F-D-FACTOR>**<F-D-PRIME>

<F-D-PRIME>::= <F-CONSTANT> \ <F-VBLE> \ <F-FCT> \
(<F-D-EXPR>) \ <ARG.IND>

<ARG.IND>::= [<P.I>]
<P.I>::= <I> SUCH THAT ITS VALUE IS BETWEEN
0 AND 20.

<F-D-EXPR> MUST CONTAIN AT LEAST ONE <ARG.IND>.

8.1.3. FORMAL FUNCTIONAL EXPRESSIONS.

<F-FCT>::= <F-MATH> \ <F-SPECIAL> \ <F-DIFF> \ <F-DEF> \
<F-UNDEF>

<F-MATH>::= <M1>[<F-EXPR>] \ <M2>[<F-EXPR>,<F-EXPR>] \
<M3>[<F-EXPR>,<F-EXPR>,<F-EXPR>]

<M1>::= EXP \ LOG \ TNH \ SIN \ COS \ ATN \ FAC

<M2>::= BIN

<M3>::= STEP

<F-SPECIAL>::= <S1>[<F-EXPR>] \ COEFF[<F-EXPR>,<F-EXPR-S>] \
ARG[<F-EXPR>,<P.I>] \ <S2>[<F-EXPR>,<F-CONST>]
<S5>[<F-EXPR>,<PAIRS>]

<S1>::= EXPAND \ CODEM \ NUM \ DENOM \ EXPON \ BASE
FLOAT \ FLOATB \ LDOP \ NARG

<S2>::= FIX \ FIXF

<F-EXPR-S>::= <F-EXPR> SUCH THAT IT IS NOT A SUM

<S5>::= SUBST \ REPLAC

<PAIRS>::= <PAIR> \ <PAIRS>,<PAIR>

<PAIR>::= <F-EXPR>,<F-EXPR>

<F-DIFF>::= DIF[<F-EXPR>,<F-VBLE>,<P.I>]

<F-UNDEF>::= <F-ID>[<ARG-LIST-7>]

<F-DEF>::= <F-ID>[<ARG-LIST-20>]

<ARG-LIST-K>::= <F-EXPR> \ <F-EXPR>,<F-EXPR> \ ...
MAXIMUM K

<F-ID>::= <ID> \ <ID>(<SUBS.LIST>)

8.1.4. FORMAL ASSIGN STATEMENT.

<F-ASSIGN>::= <F-ASG> \ <F-L-ASG> \ <F-D-ASG>

<F-ASG>::= <F-VBLE>=<F-EXPR>

<F-L-ASG>::= <F-L-VBLE> = <F-EXPR>,<F-EXPR> \

<F-L-ASG> , <F-EXPR> \

<F-L-VBLE>::= <F-VBLE>

<F-D-ASG>::= <F-ID>=<F-D-EXPR>

8.1.5. I/O STATEMENTS.

<F-READ>::= READ(<P,I>)<VBLE LIST>

<F-WRITE>::= WRITE(<P,I>,<P,I>,<P,I>)<VBLE LIST>

<VBLE LIST>::= <F-VBLE> \ <F-ID> \ <VBLE LIST>,<F-VBLE> \

<VBLE LIST>,<F-ID>

INPUT BY THE READ STATEMENT MUST HAVE THE FOLLOWING FORM:

<INPUT>::= <F-EXPR>; \ <F-D-EXPR>; \ <INPUT><F-EXPR>; \

<INPUT><F-D-EXPR>;

8.1.6. OPTION STATEMENT

<F-OPTION>::= OPTION <OPT.LIST>

<OPT.LIST>::= <OPT.TERM> \ <OPT.LIST>,<OPT.TERM>

<OPT.TERM>::= INT \ NOINT \ MFCT \ NOMFCT \

PRODEX \ NOPREX \

BASE(0) \ BASE(2) \ BASE(10) \ BASE(E) \
EXPAND(<P.I>)

8.1.7. ERASE STATEMENT.

<F-ERASE>::= ERASE <VBLE LIST>

8.1.8. DUMP STATEMENT.

<F-DUMP>::= DUMP SYMBOLS \ DUMP ALL SYMBOLS \
DUMP EXPRESSIONS \ DUMP ALL EXPRESSIONS

8.1.9. ROLOUT, SAVE AND RESET STATEMENTS.

<ROLL-OUT>::= ROLOUT <VBLE LIST>
<SAVE>::= SAVE <FILE ID>
<RESET>::= RESET <FILE ID>
<FILE ID>::= .<ELT ID> \ <FILE NAME> . <ELT ID>
<ELT ID>::= <ELT NAME> \ <ELT NAME> / <VERS NAME>

8.1.10. FORMAL STATEMENTS.

<F-STATEMENT>::= '<F-TYPE>' \ <LABEL> ' <F-TYPE> '
<LABEL>::= <P.I>
<F-TYPE>::= <F-ASSIGN> \ <F-READ> \ <F-WRITE> \

<F-OPTION> \ <F-ERASE> \ <F-DUMP> \

<ROLL-OUT> \ <SAVE> \ <RESET>

8.1.11. FORTRAN VALUED FUNCTIONS.

<FN.FCT>::= LDOP('<F-EXPR>') \ NARG('<F-EXPR>') \

IVALUE('<F-EXPR>') \ VALUE('<F-EXPR>') \

IDENT('<F-EXPR>,<F-EXPR>') \ NCOUNT(0)

9. INTERNAL REPRESENTATION OF EXPRESSIONS

THE GENERATED SYMBOLIC ALGEBRAIC EXPRESSIONS ARE REPRESENTED IN LINEAR ARRAYS. EACH ITEM OF AN EXPRESSION OCCUPIES ONE ENTRY IN THE ARRAY WHICH IS REFERENCED AS A PAIR OF WORDS, C-D, OR IN INDEXED FORM, C(I)-D(I). THERE ARE TWO EXCEPTIONS TO THIS RULE: THE RATIONAL CONSTANTS OCCUPY TWO CONSECUTIVE PAIRS WHERE THE FIRST PAIR IS USED FOR THE NUMERATOR AND THE SECOND PAIR FOR THE DENOMINATOR. THE SUBSCRIPTED VARIABLES ALSO USE TWO CONSECUTIVE PAIRS, THE FIRST PAIR IS FOR THE NAME OF THE VARIABLE, THE SECOND PAIR IS FOR THE SUBSCRIPTS.

THE EXPRESSIONS STORED IN THESE LINEAR ARRAYS ARE IN PREFIX NOTATION WHICH IS DECRIBED IN SECTION 9.2. THE MINUS AND DIVIDE OPERATORS ARE NOT REPRESENTED INTERNALLY; THE EQUIVALENT FORMS ARE DESCRIBED IN SECTION 9.1. TO ACHIEVE UNIQUE REPRESENTATION OF IDENTICAL EXPRESSIONS, AN ORDERING IS IMPOSED ON THE ARGUMENTS OF SYMMETRIC OPERATORS; THIS IS DESCRIBED IN SECTION 9.3. SECTION 9.4 DESCRIBES THE DEFINITIONS USED FOR THE FORMAL FUNCTIONS BASED ON THE INTERNAL REPRESENTATION OF EXPRESSIONS, E.G. THE NUMERATOR OF AN EXPRESSION. SECTION 9.5. DESCRIBES THE SIMPLIFICATIONS ACHIEVED WITH THE OPERATIONS OF EXPRESSIONS BASED ON THEIR INTERNAL REPRESENTATION.

9.1. TRANSFORMING MINUS AND DIVIDE OPERATORS

LET E,E1,E2 REPRESENT ARBITRARY EXPRESSIONS. THE FOLLOWING TRANSFORMATIONS ARE USED TO DELETE THE UNARY PLUS, MINUS, AND DIVIDE OPERATORS:

```
      +E => E                  (UNARY +)

      -E => (-1)*E             (UNARY -)

   E1-E2 => E1+(-1)*E2         (BINARY -)

   E1/E2 => E1*E2**(-1)        (BINARY /)
```

THESE TRANSFORMATIONS ARE APPLIED AUTOMATICALLY DURING THE INFIX TO PREFIX TRANSFORMATION.

9.2. EXPRESSIONS IN PREFIX NOTATION

THE PREFIX NOTATION OF ALGEBRAIC EXPRESSIONS IS CHARACTERIZED

BY THE FACT THAT THE OPERATORS PRECEDE THEIR ARGUMENTS. THUS THE THREE BASIC ARITHMETIC OPERATORS - ADD, MULTIPLY, AND EXPONENTIALIZE - AS BINARY OPERATORS WITH ARGUMENTS A AND B ARE AS FOLLOWS:

INFIX:	PREFIX:
A + B	+ A B
A * B	* A B
A ** B	** A B

IN THE ABOVE EXAMPLES, ALL OPERATORS WERE BINARY OPERATORS. FOR GENERAL SPECIFICATION, WE MUST ESTABLISH THE FOLLOWING NOTATION:

LET

C,C1,C2,...,CK	REPRESENT CONSTANTS
V,V1,V2,...,VL	VARIABLES
E,E1,E2,...,EM	EXPRESSIONS
F(N),F1(N),...	OPERATORS OR FUNCTIONS WITH N ARGUMENTS.

FURTHERMORE, LET A PERIOD BE USED FOR THE SEPARATION OF SYMBOLS. E.G.

+(3).V1.V2.V3

CORRESPONDS TO THE SUM OF THE 3 VARIABLES V1,V2 AND V3.

GENERALLY, THE RECURSIVE DEFINITION OF AN EXPRESSION IS AS FOLLOWS:

E := C

OR

E := V

OR

E := F(N).E1.E2.....EN (N=1,2,...)

THE FIRST ITEM OF AN EXPRESSION IN PREFIX FORM IS CALLED THE LEAD OPERATOR.

9.3. ORDERING

THE TWO ARITHMETIC OPERATORS, + AND *, ARE SYMMETRIC OPERATORS, I.E. E1+E2=E2+E1 AND E1*E2=E2*E1. THIS CAUSES A NON-UNIQUE REPRESENTATION OF OTHERWISE IDENTICAL EXPRESSIONS. FOR THE SOLUTION OF THIS PROBLEM, AN ORDERING IS IMPOSED ON THE EXPRESSIONS. THIS ORDERING IS DEFINED ON THE PREFIX FORM OF THE EXPRESSIONS, AND IT

IS APPLIED FOR THE ORDERING OF ARGUMENTS OF THE SYMMETRIC OPERATORS, + AND *. THUS, E1+E2.AND E1*E2 ARE REPRESENTED IN PREFIX FORM AS FOLLOWS

+(2).E1.E2 AND *(2).E1.E2 IF E1 < E2

+(2).E2.E1 AND *(2).E2.E1 IF E1 > E2

GENERALLY,

+(N).E1...EI...EJ...EN, *(N).E1...EI...EJ...EN

IMPLIES EI < EJ FOR I<J ,I,J=1,...,N. ONCE THE ORDERING OF EXPRESSIONS IS ESTABLISHED, THE UNIQUE REPRESENTATION OF EXPRESSIONS IS ASSURED.

THE ORDERING OF EXPRESSIONS IS DONE ACCORDING TO THEIR INTERNAL BINARY REPRESENTATION. EVERY ITEM OF AN EXPRESSION IS REPRESENTED BY TWO INTEGER FIELDS, TYPE FIELD, AND DATA FIELD, ITYP(I), AND D(I). THE RELATION OF TWO EXPRESSIONS,

E1 = S1.S2...SN, E2 = T1.T2...TM,

IS DETERMINED BY COMPARING ITS INTERNAL FIELDS,

ITYP(S1).D(S1).ITYP(S2).D(S2)...ITYP(SN).D(SN) AND
ITYP(T1).D(T1).ITYP(T2).D(T2)...ITYP(TN).D(TN),

FROM LEFT TO RIGHT. THE FIRST NON-EQUAL FIELD DETERMINES THE RELATION OF THE EXPRESSIONS. I.E.

ITYP(SJ) = ITYP(TJ) AND D(SJ) = D(TJ) FOR J=1,...,K-1
AND ITYP(SK) < ITYP(TK)
IMPLIES E1 < E2

OR

ITYP(SJ) = ITYP(TJ) AND D(SJ) = D(TJ) FOR J=1,...,K-1
AND ITYP(SK)=ITYP(TK) AND D(SK) < D(TK)
IMPLIES E1 < E2.

WHEN ALL FIELDS ARE EQUAL THEN THE TWO EXPRESSIONS ARE IDENTICAL. SINCE THE TYPE FIELD OF THE FIRST ITEM CORRESPONDS TO THE LEAD OPERATOR OF AN EXPRESSION, THE PRIMARY SORTING OF THE EXPRESSIONS IS BY THEIR LEAD OPERATORS.

THE INTERNAL REPRESENTATION OF THE ITEMS OF EXPRESSIONS IS LISTED IN TABLE I IN APPENDIX A.

9.4. DEFINITION OF FORMAL FUNCTIONS

THIS SECTION DEFINES THE FORMAL FUNCTIONS (CODEM,COEFF,NUM,DENOM, EXPON,BASE) BASED ON THE PREFIX REPRESENTATION OF THEIR ARGUMENTS. THE NOTATION USED IS THE SAME AS IN THE PREVIOUS SECTION, NAMELY

C - FOR CONSTANTS
V - FOR VARIABLES
E - FOR EXPRESSIONS
F(N)- FOR OPERATORS, FUNCTIONS WITH N ARGUMENTS

9.4.1. NUM, DENOM

WHENEVER THE ARGUMENTS OF THESE FUNCTIONS CONTAIN A DUMMY VARIABLE OF AN UNDEFINED FUNCTION, THEY ARE NOT EVALUATED.

THE NUM AND DENOM FUNCTIONS GIVE THE NUMERATOR AND DENOMINATOR OF AN EXPRESSION E, RESPECTIVELY. OBVIOUSLY,

NUM[E] * (DENOM[E]**-1) = E

IN INFIX NOTATION, THE DEFINITION OF THE NUMERATOR/DENOMINATOR OF AN EXPRESSION DEPENDS ON ITS LEAD OPERATOR. LET

E = F(N).E1.E2...E^N

BE THE EXPRESSION, THEN

IF F ≠ *,** I.E. THE EXPRESSION IS NOT A PRODUCT OR EXPONENTIAL EXPRESSION, THEN
NUM[E] = E AND DENOM[E] = 1

IF F = ** I.E. THE EXPRESSION IS AN EXPONENTIAL EXPRESSION WHICH MUST BE IN THE FOLLOWING FORM:
E = **(2).E1.E2
WHERE E1 IS THE BASE AND E2 IS THE EXPONENT. LET E2 HAVE THE FOLLOWING FORM
E2 = G(M).E21.E22...E2M.
THEN HAVE THE EXPONENT E2 SEARCHED FOR NEGATIVE CONSTANTS:
IF G(M)=C SUCH THAT C<0 THEN
NUM[E]=1 DENOM[E]=E**(-1)
IF G(M)=*(M) THEN
IF E21=C SUCH THAT C<0 THEN
NUM[E]=1 DENOM[E]=E**(-1)
OTHERWISE
NUM[E]=E DENOM[E]=1
IF G(M)=+(M) THEN
E=*(M).**(2).E1.E21. ... .**(2).E1.E2M

AND THE FACTORS FOR THE NUMERATOR/
DENOMINATOR ARE DETERMINED BY THE
PREVIOUS TWO CONDITIONS.

IF F = * I.E. E IS A PRODUCT, THEN ALL FACTORS OF E, E1,...,EN, ARE CHECKED FOR THEIR NUMERATORS AND DENOMINATORS BY THE PREVIOUS CONDITIONS AND FINALLY:

NUM[E]= *(N).NUM[E1].NUM[E2]...NUM[EN]
DENOM[E]= *(N).DENOM[E1]...DENOM[EN]

9.4.2. CODEM

FUNCTION CODEM PUTS THE ARGUMENT EXPRESSION IN COMMON DENOMINATOR FORM. IF THE EXPRESSION IS NOT A SUM, I.E. ITS LEAD OPERATOR IS NOT +, THEN CODEM DOES NOT CHANGE ITS FORM. WHEN THE EXPRESSION E IS A SUM,

E = +(N).E1.E2...EN,

THEN THE DENOMINATORS OF THE SUBEXPRESSIONS ARE DETERMINED:

DENOM[E1],...,DENOM[EN]

AFTER DISCARDING THE DENOMINATORS WHICH HAVE VALUES IDENTICAL TO 1, THE LEAST COMMON MULTIPLE IS DETERMINED AS THE PRODUCT OF CERTAIN DENOMINATORS FROM THE ABOVE LIST:

ED = *(M).DENOM[EI1]...DENOM[EIM] (M.LE.N)

THE DISCARDED DENOMINATORS ARE THOSE FOR WHICH THERE IS ANOTHER DENOMINATOR WITH THE SAME BASE AND A LARGER EXPONENT, I.E.

DENOM[EL] = **(2).EL1.EL2

IS DISCARDED IF THERE IS A
DENOM[EIK] = DENOM[EJ] = **(2).EJ1.EJ2

SUCH THAT

EL1 = EJ1 AND EITHER EL2=C1 < C2=EJ2
OR EL2=*(K).C1.EE2...EEK AND
EJ2=*(K).C2.EE2...EEK AND
C1 < C2

AFTER THE COMMON DENOMINATOR, ED, IS DETERMINED, THE ORIGINAL EXPRESSION, E, IS MULTIPLIED BY IT TERM BY TERM, AND THE RESULT IS MULTIPLIED BY THE COMMON DENOMINATOR RAISED TO THE -1 POWER:

DENOM[E] =

```
*(2).+(N).*(2).E1.ED...*(2).EN.ED.**(2).ED.-1
```

9.4.3. EXPON, BASE

WHENEVER THE ARGUMENTS OF THESE FUNCTIONS CONTAIN A DUMMY VARIABLE OF AN DEFINED FUNCTION, THEY ARE NOT EVALUATED.

FUNCTIONS EXPON AND BASE RETURN THE EXPONENT OR BASE OF AN EXPRESSION E, RESPECTIVELY. WHEN THE LEAD OPERATOR OF E IS NOT AN EXPONENTIAL OPERATOR, THEN EXPON[E]=1 AND BASE[E]=E. WHEN THE LEAD OPERATOR OF E IS **, E = **(2).E1.E2, THEN EXPON[E]=E2, BASE[E]=E1.

9.4.4. COEFF

WHENEVER THE ARGUMENT OF THIS FUNCTION CONTAINS A DUMMY VARIABLE ON A DEFINED FUNCTION, IT IS NOT EVALUATED.

COEFF[E1,E2] RETURNS THE COEFFICIENT OF THE EXPRESSION OF E2 IN E1. E2 MUST BE EITHER A VARIABLE OR AN EXPRESSION WITH LEAD OPERATOR **, MATHEMATICAL, DIFFERENTIAL, OR UNDEFINED FUNCTION, I.E. E2 MAY NOT BE A CONSTANT, SUM OR PRODUCT.

THERE ARE 3 CASES DEPENDING ON THE LEAD OPERATOR OF E1:

1. E1 IS NEITHER A SUM NOR A PRODUCT, I.E. THE LEAD OPERATOR OF E1 IS NEITHER + NOR *, THEN:

```
COEFF[E1,E2] = 1    IF  E1=E2
COEFF[E1,E2] = 0    IF  E1≠E2
```

2. E1 IS A PRODUCT, I.E. THE LEAD OPERATOR OF E1 IS *:

```
E1 = *(N).E11.E12...E1N
```

```
THEN      COEFF[E1,E2]=*(N-1).E11...E1K.E1J...E1N,
                                  K=I-1,J=I+1

                                  IF  EI=E2

          COEFF[E1,E2] = 0        OTHERWISE
```

3. E1 IS A SUM, I.E. THE LEAD OPERATOR OF E1 IS +:

```
E1 = +(N).E11.E12...E1N
```

THEN THE RESULT IS THE SUM OF THE COEFFICIENTS IN THE INDIVIDUAL TERMS:

```
COEFF[E1,E2] =
```

```
+(N).COEFF[E11,E2].COEFF[E12,E2]...COEFF[E1N,E2]
```

9.5. SIMPLIFICATIONS.

9.5.1. AUTOMATIC SIMPLIFICATION.

CONSTANTS:

ARITHMETIC OPERATIONS BETWEEN CONSTANTS ARE ALWAYS EVALUATED PRODUCING A NEW CONSTANT WHOSE TYPE DEPENDS ON THE OPERATION AND THE CONSTANTS OPERATED UPON AS FOLLOWS:

I = INTEGER
R = RATIONAL
F = FLOATING POINT

ZERO IS ALWAYS TYPE I
RATIONAL CONSTANTS ARE ALWAYS SIMPLIFIED

ADDITION:

```
I + I => I
I + R => R
I + F => F     UNLESS IT IS ZERO
R + R => R     UNLESS IT SIMPLIFIES TO INTEGER
R + F => F     UNLESS IT IS ZERO
F + F => F     UNLESS IT IS ZERO
```

MULTIPLICATION:

```
I * I => I
I * R => R     UNLESS IT SIMPLIFIES TO INTEGER
I * F => F     UNLESS IT IS ZERO
R * R => R     UNLESS IT SIMPLIFIES TO INTEGER
R * F => F
F * F => F
```

EXPONENTIALIZATION:

ZERO IF BASE IS ZERO
INTEGER ONE IF BASE IS ONE
INTEGER ONE IF EXPONENT IS ZERO

```
I ** I => I
          R    IF EXPONENT IS NEGATIVE
I ** R => F
          ERROR IF BASE IS NEGATIVE AND DENOMINATOR OF
                EXPONENT IS EVEN
I ** F => F
          ERROR IF BASE IS NEGATIVE
R ** I => R
R ** R => F
          ERROR IF BASE IS NEGATIVE AND DENOMINATOR OF
```

```
                    EXPONENT IS EVEN
  R ** F => F
            ERROR IF BASE IS NEGATIVE
  F ** I => F
  F ** R => F
            ERROR IF BASE IS NEGATIVE AND DENOMINATOR OF
                    EXPONENT IS EVEN
  F ** F => F
            ERROR IF BASE IS NEGATIVE
```

NON-CONSTANTS:

ARITHMETIC OPERATIONS:
E = ARBITRARY EXPRESSION, C = CONSTANT

```
  0 + E => E
  1 * E => E              1.0 * E => E
  0 ** E => 0
  1 ** E => 1             1.0 ** E => 1
  E ** 0 => 1
  E ** 1 => E             E ** 1.0 => E
  C1 * E + C2 * E => ( C1 + C2 ) * E
  ( E1 ** E2) ** E3 => E1 ** ( E2 * E3 )
  ( E ** E1 ) * ( E ** E2 ) => E ** ( E1 + E2 )
```

MATHEMATICAL FUNCTIONS:

```
  EXP[ LOG[ E ]] => E
  LOG[ EXP[ E ]] => E
  LOG[ E1 ** E2 ] => E2 * LOG[ E1 ]
  SIN[ -1 * E ] => -1 * SIN[ E ]
  TNH[ -1 * E ] => -1 * TNH[ E ]
  ATN[ -1 * E ] => -1 * ATN[ E ]
  COS[ -1 * E ] => COS[ E ]
```

9.5.2. OPTION CONTROLLED SIMPLIFICATION.

OPTION: EXPAND(K) OR FUNCTION: EXPAND
(N, K ARE POSITIVE INTEGERS)

```
  E1 * ( E2 + E3 ) => E1 * E 2 + E1 * E3
  ( E1 + E2 ) ** N => E1 ** N +
                      BIN[N,1] * E1**(N-1) * E2 +
                                  . . .
                      BIN[N,N-1] * E1 * E2**(N-1) +
                      E2 ** N                  ( IF  K > N-1 )
  EXP[ E1 + LOG[ E2 ]] => E2 * LOG[ E1 ]
  EXP[ E1 * LOG[ E2 ]] => E2 ** E1
  LOG[ E1 * E2 ] => LOG[ E1 ] + LOG[ E2 ]
```

OPTION: BASE(K) WITH K= 2 OR 10

```
  C ** E => K ** ( E * LOG[C]/LOG[K] )        ( IF C > 0 )
```

```
OPTION: BASE(E)
     C ** E => EXP[ E * LOG[C] ]                    ( IF C > 0 )

OPTION: PRODEX  OR  FUNCTION: PRODEX
     ( E1 * E2 ) ** E3 => ( E1 ** E3 ) * ( E2 ** E3 )

OPTION: MFCT
     EXP[ C1 ] => C2
     LOG[ C1 ] => C2      ( C1 > 0 )
     TNH[ C1 ] => C2
     SIN[ C1 ] => C2
     COS[ C1 ] => C2
     ATN[ C1 ] => C2

OPTION: INT          ( I IS NON-NEGATIVE INTEGER )
     FAC[ I ] => 1 * 2 * ... * I
     BIN[ I1,I2 ] => (I1/1) * ((I1-1)/2) * ... * ((I1-I2+1)/I2
                                                ( I1 > I2-1 )
     BIN[ E,0 ] => 1
     BIN[ E,1 ] => E
     BIN[ E,E ] => 1
     STEP[ C1,C2,C3 ] => 1 IF  C1=< C2 < C3  OR C1=C2=C3
                         0 OTHERWISE
```

10. APPENDIX A

TABLE I.

INTERNAL REPRESENTATION OF EXPRESSIONS.

ITYP	D(I)	REPRESENTS
0	INTEGER	A SIGNED INTEGER AS CONSTANT
1	INTEGER	A SIGNED INTEGER AS THE NUMERAT. OF A RATIONAL CONSTANT.
2	POSITIVE INTEGER	A POSITIVE NON-ZERO INTEGER AS THE DENOMINATOR OF A RATIONAL CONSTANT. NUMERATOR AND DENOMINATOR ARE ALWAYS APPEARING CONSECUTIVELY.
3	FLOATING-POINT NUMBER	A REAL CONSTANT.
5	POSITIVE INTEGER=I	I'TH ARGUMENT OF A FUNCTION DEFINED BY AN EXPRESSION.
6	ALPHANUMERIC NAME	NON-SUBSCRIPTED VARIABLE WHOSE VALUE IS ITS NAME, D(I).
7	ALPHANUMERIC NAME	SUBSCRIPTED VARIABLE WHOSE VALUE IS ITS NAME, D(I), WITH ITS SUBSCRIPTS DEFINED IN THE NEXT TERM.
8	INTEGER IN FIELD 0-8 FIELD 9-35 = 0	SUBSCRIPT WORD WITH 1 SUBSCRIPT
9	INTEGER FIELDS 0-8 AND 9-17. FIELD 18-26 = 0	SUBSCRIPT WORD WITH 2 SUBSCRIPTS
10	INTEGER FIELDS 0-8, 9-17 AND 18-26, FIELD 27-35 = 0	SUBSCRIPT WORD WITH 3 SUBSCRIPTS

TABLE I. (CONTINUED)

11	INTEGER FIELDS 0-8, 9-17, 18-26 AND 27-35	SUBSCRIPT WORD WITH 4 SUBSCRIPTS SUBSCRIPTED VARIABLE IS ALWAYS FOLLOWED BY A SUBSCRIPT WORD.
16	POSITIVE INTEGER H1 = FIELD 0-17 H2 = FIELD 18-35	A COMMA (,), WHERE H1 IS THE NUMBER OF FOLLOWING EXPRESSIONS. H2=0 IF THE EXPRESSIONS ARE IN THE CORE. H2=INDEX IF THE EXPRESSIONS ARE ON DRUM.
17	NOT USED	REPRESENTS (, IT IS USED ONLY DURING TRANSLATION FROM INFIX TO PREFIX
18	POSITIVE INTEGER GREATER THAN 1	ADD (+) OPERATOR WHERE D(I) IS THE NUMBER OF FOLLOWING SUB-EXPRESSIONS TO BE ADDED.
19	POSITIVE INTEGER GREATER THAN 1	MULTIPLY (*) OPERATOR WHERE D(I) IS THE NUMBER OF FOLLOWING SUB-EXPRESSIONS TO BE MULTIPLIED.
19	-2	DIVISION /, USED ONLY DURING TRANSLATION FROM INFIX TO PREFIX
20	2	EXPONENTIALIZATION SUCH THAT THE FOLLOWING TWO SUB-EXPRESSIONS REPRESENT THE BASE AND EXPONENT, RESPECTIVELY.

TABLE I. (CONTINUED)

21	INTEGER FIELDS 0-17 AND 18-35	MATHEMATICAL FUNCTIONS WHERE FIELDS 18-35 REPRESENTS THE NUMBER OF FOLLOWING SUB-EXPR. AS ARGUMENTS AND FIELD 0-17 IS AN IDENTIFIER INTEGER AS FOLLOWS		
			32+128	EXP
			33+128	LOG
			34+128	TNH
			35+128	SIN
			36+128	COS
			37+128	ATN
			50+128	FAC
			51+256	BIN
			52+384	STEP
		THE FOLLOWING IDENTIFIER INTEG. APPEAR ONLY DURING TRANSLATION: (EXCEPT THE # ONES)		
			64+128	EXPAND
			65+128	CODEM
			66+256	COEFF #
			67+128	NUM #
			68+128	DENOM #
			69+128	EXPON #
			70+128	BASE #
			71+128	ARG #
			72+640	SUBST
			73+640	REPLAC
			75+128	PRODEX
			76+128	FLOAT
			77+128	FLOATB
			78+256	FIX
			79+256	FIXE
			80+128	LDOP
			81+128	NARG

TABLE I. (CONTINUED)

22	FIELD 0-17: EXPRESSION POINTER FIELD 18-35: NUMBER OF ARGUMENTS	DEFINED FUNCTION, USED ONLY DURING TRANSLATION FROM INFIX TO PREFIX
23	2	DIFFERENTIATION OF A FUNCTIONAL EXPRESSION BY A VARIABLE. IT IS FOLLOWED BY A SUBSCRIPTED OR NONSUBSCRIPTED VARIABLE AND BY AN UNDEFINED FUNCTION OR OTHER DIFFERENTIAL OPERATOR.
25-31	ALPHANUMERIC NAME	AN UNDEFINED FUNCTION WHOSE NAME IS D(I). IT IS FOLLOWED BY ITYP(I)-24 SUB-EXPRESSIONS AS ITS ARGUMENTS